PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós-graduação em Engenharia Elétrica

Alysson Ribeiro da Silva

Creative Agent Reasoning through Adaptive Neural Networks

Belo Horizonte
2017

Alysson Ribeiro da Silva

Creative Agent Reasoning through Adaptive Neural Networks

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica from the Pontifícia Universidade Católica de Minas Gerais, as a partial requirement in order to earn a Master of Science degree in Electrical Engineering.

Supervisor: Luís Fabrício Wanderley Góes
Co-supervisor: Carlos Augusto Paiva da Silva Martins

Belo Horizonte
2017

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Silva, Alysson Ribeiro da
S586c Creative agent reasoning through adaptive neural networks / Alysson Ribeiro da Silva. Belo Horizonte, 2017.
251 f. : il.

Orientador: Luís Fabrício Wanderley Góes
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica

1. Redes neurais (Computação). 2. Pensamento criativo. 3. Algorítmos computacionais. 4. Inteligência artificial. 5. Jogos eletrônicos. I. Góes, Luís Fabrício Wanderley. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 681.3.091

Alysson Ribeiro da Silva

Creative Agent Reasoning through Adaptive Neural Networks

Thesis presented to the Programa de Pós-graduação em Engenharia Elétrica from the Pontifícia Universidade Católica de Minas Gerais, as a partial requirement in order to earn a Master of Science degree in Electrical Engineering.

Luís Fabrício Wanderley Góes, Ph.D. - PUC Minas (Supervisor)

D.r Felipe Domingos da Cunha - PUC Minas

D.r Luiz Chaimowicz - UFMG

Belo Horizonte, 17 November 2017

I would like to dedicate this work in memory of Eva Maria da Silva since she was a very important person in my life. I also would like to dedicate this work to my mom and dad since both take care of me until to this date, providing the necessary environment, incentive, emotional support and love that allows me to pursue my dreams visioning a better world. I would like to dedicate this work to my university colleagues that were unable to graduate due to social economical problems. I would like to dedicate this work to Camila Guedes Silveira since she was at my side at every moment during last years. In addition, I also dedicate this work to Carlos Augusto De Paiva Silva Martins, Luís Fabrício Wanderley Góes, Alexei Manso Correa Machado and Max do Val Machado, professors at PUC Minas, since they provided me valuable advices and helped me during my life. In special, I would like to dedicate this work to any who not blesses with basic living conditions, any who, that unfortunately, is obligated to interact with human corruption, any who, that unfortunately, needs to deal with jealous, envy, angry and lies from others, and any who that does not had the opportunity to conduct research in any field of study, since those conditions, among others, could possibly deprive the world from great scientists, and more important, great human beings.

ACKNOWLEDGEMENTS

I acknowledge Alessandro Ribeiro da Silva for helping me in understand the minefield navigation simulation system and for introducing me the Java Native Interface, since it helped me in replicating the original Adaptive Resonance Associative Map research conducted by A. H. Tan. and in integrating Java with c++/CUDA. In addition, I acknowledge Celso Renato França França for suggesting that the Bayesian surprise model could be used in order to compute the novelty of an idea. I also acknowledge, Luís Fabrício Wanderley Góes for providing a Tesla Graphics Processing Unit to conduct experiments and for suggesting that the Honing Theory would possibly be able to generate creative ideas. Furthermore, I acknowledge Camila Guedes Silveira for helping me, during moments of emotional need, on the development of the time complexity analysis for all deployed systems. Finally, I acknowledge FAPEMIG and CAPES for the granted scholarships although both did not provided personal income in terms of salary.

“The original question, "Can machines think?"
I believe to be too meaningless to deserve discussion.”
Alan Mathison Turing (TURING, 1995)

ABSTRACT

In recent years, neural networks have been used successfully to control agents. Nevertheless, little effort is dedicated on the deployment of creative behavior on them. In that context, our main objective, divided into three parts, is to propose and develop methods to deploy human creativity into an agent's reasoning process. The first part envisioned the creation of a computational model of The Honing Theory of creativity, that has never been attempted before, which led us to the formulation of a computational model of it. This model was deployed in a system we called HoningStone to generate creative card combos for a digital collectible card game called HearthStone. Our results showed that HoningStone could generate card combos that are more creative than the ones generated by a greedy randomized algorithm driven by the same creativity metric. The second part envisioned to enable an agent to build an emergent solution, not only to build combos but to generate full strategies which at first are just efficient, not creative. In this new scenario, the search space is even larger due to the number of cards and actions combinations and also due to randomness. We then proposed an Adaptive Neural Network for emergent agent development and control using semantic information. This proposal was deployed on Hearthbot, an autonomous agent we created that plays Hearthstone. Results show that it achieved an average win rate performance of 80% against the Monte Carlo Tree Search. The last part envisioned to make HearthBot also creative and more efficient in terms of win rate performance, where the Honing Theory was fit within the natural learning process of the Adaptive Resonance Theory. It led us to the proposal and creation of the Honing Adaptive Resonance Process. It is composed of Adaptive Neural Networks we design to handle creative thinking. Results show that the creative Hearthbot agents surpassed the performance of the original, and it achieved a win rate above 90%, on average, against the Monte Carlo Tree Search and above 60%, on average, when playing against the handcrafted algorithm based on greedy local decisions.

Keywords: Adaptive Neural Network. Creative Thinking. The Honing Theory. Adaptive Resonance Theory. HearthStone.

RESUMO

Nos últimos anos, as redes neurais foram utilizadas com sucesso para controlar os agentes. No entanto, pouco esforço é dedicado na implantação de comportamento criativo nos mesmos. O principal objetivo desta pesquisa, dividido em três partes, é propor e implantar métodos capazes de implantar a criatividade humana no processo de raciocínio de agentes. A primeira parte desta pesquisa visa a criação de um modelo computacional da Teoria do Aprimoramento, que nunca foi tentado antes, o que nos levou à formulação de um modelo computacional capaz de representa-la. Este modelo foi implantado em um sistema que chamamos HoningStone para gerar combos de cartas criativos para um jogo digital de cartas colecionáveis chamado HearthStone. Nossos resultados mostraram que HoningStone foi capaz de gerar combos de cartas que são mais criativos do que os gerados por um algoritmo guloso conduzido pela mesma métrica de criatividade. A segunda parte visa permitir que um agente contrua uma solução emergente, não só para criar combos, mas também para gerar estratégias completas que em primeiro lugar são apenas eficientes, não criativas. Neste novo cenário, o espaço de busca é ainda maior devido ao número de combinações de cartas e ações e também devido à aleatoriedade. Em seguida, propusemos uma Rede Neural Adaptativa para desenvolvimento e controle de agentes emergentes usando informações semânticas. Esta proposta foi implantada no Hearthbot, um agente autônomo que criamos que é capaz de jogar Hearthstone. Os resultados mostram que o mesmo alcançou um desempenho médio da taxa de vitórias perto de 80% contra o Monte Carlo Tree Search. Na última parte, esta pesquisa visa tornar o HearthBot também criativo e mais eficiente em termos de desempenho da taxa de vitória, onde a Teoria do Aprimoramento foi incorporada no processo de aprendizagem natural da Teoria da Ressonância Adaptativa. Isso nos levou à proposta e criação do Processo de Aprimoramento de Ressonância Adaptativa. É composto de Redes Neurais Adaptativas que projetamos para lidar com o pensamento criativo. Tal solução foi implantada em agentes que chamamos de HearthBots Criativos. Os resultados mostram que o desempenho dos mesmos superaram suas versões não criativas, e alcançaram uma taxa de vitórias acima de 90%, em média, contra o Monte Carlo Tree Search e acima de 60%, em média, quando jogando contra uma heurística gulosa baseado em construções aleatorias.

Palavras-chave: Redes Neurais Adaptativas. Pensamento Criativo. Teoria do Aprimora-

mento. Teoria da Ressonância Adaptativa. HearthStone.

LIST OF FIGURES

LIST OF TABLES

LIST OF CHARTS

ALGORITHMS LIST

LIST OF ABBREVIATIONS AND ACRONYMS

ANN	Adaptive Neural Network
ARAM	Adaptive Resonance Associative Map
ART	Adaptive Resonance Theory
ARTMAP	Adaptive Resonance Theory Map
BC-Greedy	Board Control Greedy
FALCON	Fusion Architecture for Learning COgnition and Navigation
GRASP	Greedy Randomized Adaptive Search Procedure
HARP	Honing Adaptive Resonance Process
MCTS	Monte Carlo Tree Search
POMDP	Partially Observable Markov Decision Process
UAM-ANN	Unstructured Areas Multi-channel Adaptive Neural Network

CONTENTS

1 INTRODUCTION

In the last years, neural networks have been used successfully in order to control agents (TENG; TAN, 2015; MNIH et al., 2015; SILVER et al., 2016; MIYASHITA et al., 2017). However, little effort is dedicated on the deployment of creative behavior on them. Most of the solutions that deploy creative behavior on automated systems, such as agents, are related to artistic fields (KOWALIW; DORIN; MCCORMACK, 2012; DIPAOLA, 2014; KIM; CHO, 2000; CHENG; LIU, 2008; ZHANG; YANG, 2013; VARSHNEY et al., 2013; AMORIM et al., 2017). The main challenge, from the creative agent controlling perspective, is the creation of a system that is able to grow by itself in a fast and stable way to perform valuable actions, and eventually to develop creative behavior. This is a hard task, since controlling an agent by itself is not trivial and it involves modeling and extracting data from its environment and storing it in a way that it can be used to decide on how to behave properly. Agent control is not only about how to behave, in terms of reason, but also in terms of motor control and skill, which turns the problem even harder to solve. Some attempts on how to deploy creative behavior can be seen in the literature and are typically based on incorporating, in agents, aspects of how creativity emerges on the human brain (AUGELLO et al., 2017; FITZGERALD; GOEL; THOMAZ, 2017; AGUILAR; PéREZ, 2017). However, none of the observed solutions deploy a creative agent, based fully on well-established theories on how the human brain works and how creativity emerges from its mechanism, that can grow and develop by itself in a fast and stable way.

1.1 Agent reasoning through time

When dealing with reasoning, an agent needs to concern on selecting actions that will enable it to complete its objectives. One of the biggest problems, when selecting actions to perform, is the lack of a way in simulating time. For example, if an agent could realize that its future and past actions will impact in its overall performance, thus it can decide in selecting an appropriate set of actions to be performed on the present. Time is usually simulated in computer programs as a *Discrete Event Simulation* (HENRIKSEN et al., 1986), where the concept of a *Time Flow*, set of discrete time steps, is used to make agents understand it by a concise and coherent way. Action selection is typically treated as a search problem, where given a set of possible actions, an agent needs to find a

set of future ones that will maximize its performance. To help in performing this search process, the *Time Flow* mechanism is typically represented by a Partially Observable Markov Decision Process (POMDP), where a symbolic network is used to represent how bad or good an action is at a determined time step from a *Time Flow*.

There are two major branches when dealing with the time when performing reasoning. The first one is related to how to represent and acquire information from a *Time Flow*. The second branch is related on how to search on the represented information in order to maximize an agent performance over time. These two branches can be handled by two main models on which an autonomous agent, or bot, can be built upon Symbolic and Emergent (WENG, 2012a). The symbolic solutions use data structures, algorithms, heuristics and handcrafted non-adaptive behaviors explicitly based on human expertise. In contrast, the emergent ones commonly use machine learning for designing autonomous agents. In particular, the emergent solutions can utilize paradigms like supervised learning, that learns from observations, reinforcement learning, that learns from the interaction with an existing environment or non-supervised learning, that learns without the intervention of a specialist (WANG; TAN, 2015).

Typically, symbolic solutions deploy agents that are based on the Monte Carlo Tree Search (MCTS) (WARD; COWLING, 2009a) and its variants with min-max heuristic and alpha-beta pruning (YANNAKAKIS; TOGELIUS, 2015; BAIER; WINANDS, 2015). Most of the symbolic and emergent solutions to control agents come from robotics, and they have being developed since 1970's (CARPENTER; GROSSBERG; REYNOLDS, 1991; WENG; LUCIW; ZHANG, 2013; CELIKKANAT; ORHAN; KALKAN, 2015; TAN, 2004) with the assistance of neural networks (YANNAKAKIS; TOGELIUS, 2015) and knowledge-based systems (BASILE et al., 2016; MATUSZEK et al., 2006; SAFFIOTTI et al., 2008; TENORTH; BEETZ, 2009). These networks can range from Adaptive Resonance Theory (ART), first proposed by (CARPENTER; GROSSBERG, 1988) and further extended as its fuzzy counterparts (CARPENTER; GROSSBERG; ROSEN, 1991; CARPENTER; GROSSBERG; REYNOLDS, 1991; TAN, 1995), to fully brain-like structures such as the *Temporal Context Machine* created by (WENG; LUCIW; ZHANG, 2013) and even deep learning neural networks (MIYASHITA et al., 2017; ZHAO et al., 2016; SILVER et al., 2016; SUGIMOTO et al., 2015; STEEG; DRUGAN; WIERING, 2015).

Deep learning approaches are used to control agents and are successfully used to this date. For instance, the work proposed by (SILVER et al., 2016) was the first emergent

agent that was able to win against a professional GO game player. Recently, all approaches that have been using deep learning, such as (SILVER et al., 2016; MIYASHITA et al., 2017; ZHAO et al., 2016; KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017), have implemented a mechanism, from reinforcement learning, called Q-learning. This technique optimizes a set of actions that should be taken by an agent when performing under a POMDP. However, deep learning approaches tend to be slower in terms of processing time, for example, the agent proposed by (SILVER et al., 2016) is controlled by a massive cluster of Central Processing Units and graphics processing units. Besides that, deep learning approaches are typically used to classify patterns obtained from images, since it mimics the image filtering process in its convolution layers, thus they are more suitable in interpreting what an agent is seeing than in performing a reasoning process under a POMDP.

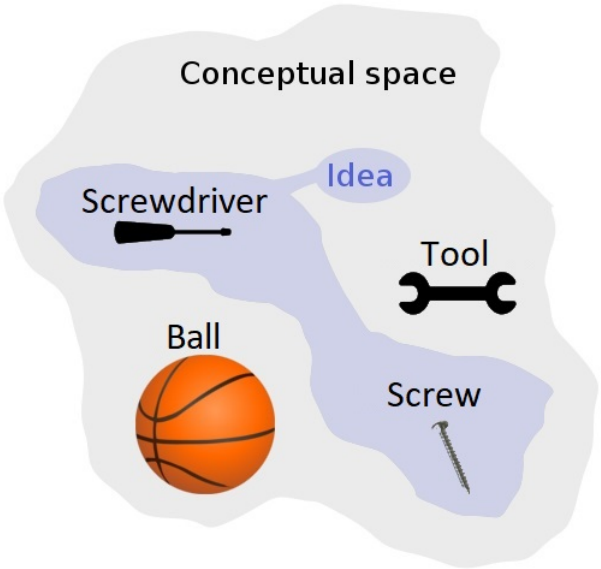
In contrast to deep learning approaches, there is some brain like structures able to perform reasoning. For instance, The ART (GROSSBERG, 1989), that explains how the human brain achieves pattern recognition, assembles neural networks that can be used to perform complex reasoning with raw stimulus and also semantic information. To this date, the most common form of implementing the ART theory is based on a neural network called Adaptive Resonance Theory Map (ARTMAP)(CARPENTER; GROSSBERG; REYNOLDS, 1991). They are from a family of neural networks called Adaptive Neural Network (ANN), that can change in size by altering its structural composition. With this mechanism, they seem suitable to represent partially observed contexts, to perform supervised and unsupervised learning. Most recent implementations of the ART, such as the fuzzy Adaptive Resonance Associative Map (ARAM) (TENG; TAN, 2015), enable the use of supervised, unsupervised and reinforcement learning within the same structure.

The ART family of ANN seems suitable to perform reasoning. However, to support their POMDP representativeness, a cognition architecture called Fusion Architecture for Learning COgnition and Navigation (FALCON) (TAN, 2004) is used. With the FALCON architecture, an agent can learn about its surroundings from scratch by observing and experiencing. Furthermore, the FALCON architecture takes advantage of the fast stable mechanism of the ART system to deal with information, thus letting agents learn and perform at the same time in real time. Besides the ability in simulating a POMDP, none of the presented solutions handle creativity and are purely used to optimize an agent's performance.

1.2 Computational Creativity for automated systems

Creative thinking is a human quality and it is defined as the capacity to express valid solutions by non-usual means. This concept was created exclusively by human cognitive skills, where its capability in handling problems can vary from fields such as politics and even science (RITCHIE, 2001). Creativity is classified into two main types: P-Creative (psychological - generate ideas with local knowledge); and H-Creative (historical - generate ideas with all available knowledge). Both types work inside a conceptual space, where concepts or symbols are represented. According to psychology, this conceptual space is where ideas can be found made by concepts. An example of a conceptual space is depicted in Figure 1, where it is composed of the concepts *screwdriver*, *screw*, *tool* and *ball*. An example of an idea that can be generated from the shown conceptual space is the fusion of the *screwdriver* with the *screw* illustrated in Figure 1, where this idea can represent that the selected tool can be used to handle a screw. Ideas can be easily generated, but the main problem when generating them is the lack of a model to measure how creative that idea really is.

Figure 1 – Creative Thinking conceptual space and idea representation.



Source: By the Author.

Measuring creativity is a hard problem since a consensual definition of what creativity really is does not exist yet, but it is speculated that its nature is related to create ideas that are novel and has value, what usually involves human feelings such as surprise,

expectation, utility and motivation (RITCHIE, 2001; SCHMIDHUBER, 2010). On the other hand, there are theories on how creativity works, such as Incubation and Conceptual Blending. The most promising theory is called The Honing Theory, which describes creativity based on recent advances in the understanding of the human brain(GABORA, 2010a) in harmony with the foundations where lies the ART networks.

Most of the solutions that are observed in the literature deploy creative systems that are able to generate artifacts that can be considered creative. In a diversity of fields, such as artistic painting (KOWALIW; DORIN; MCCORMACK, 2012; DIPAOLA, 2014), fashion (KIM; CHO, 2000; ZHANG; YANG, 2013) and even culinary (VARSHNEY et al., 2013; AMORIM et al., 2017). All deployed solutions use some sort of evolutionary algorithm or heuristics that are able to generate randomized constructions in order to represent some aspects of the theories of creative thinking. Additionally, research, as the one conducted by (JUNIOR et al., 2016), tries to represent creativity as a single equation that can be optimized and used to evaluate the creativity level of generated artifacts.

Few works on creativity were conducted in the field of agent controlling, where most of the solutions address topics such as dancing and social interactions (AGUILAR; PéREZ, 2017; AUGELLO et al., 2017; FITZGERALD; GOEL; THOMAZ, 2017). Even though machine creativity being showcased at a variety of applications, there seems to be a clear gap with respect to emergent solutions that are used to provide the means for an agent to demonstrate creativity. There is also a lack of solutions that employ computational models of the creative theories that can be used to generate solutions when performing through a POMDP. Furthermore, there are not current models that encompass creative thinking theories and adaptive resonance theories, in order for an agent to perform within a *Time Flow*.

1.3 Main objective

To equip a machine with the ability to be creative is a two-fold problem since defining what is creativity is hard but also unveiling what are the mechanisms inside the brain that promotes it is even harder. In the first front, much progress has been made in the field of Computational Creativity, in which scientists now tend to agree that a creative idea has to be novel but also useful. Quite a few general purpose metrics have been proposed to evaluate novelty, such as the Bayesian surprise, and also many

context-specific ones to measure usefulness in art, music, culinary, games. Recently, even a context-independent metric to measure both novelty and usefulness together, that is creativity, called Regent-Dependent Creativity (JUNIOR et al., 2016) has been proposed and validated on some different fields. This metric was partially inspired by one of the contributions of this thesis, which we will discuss shortly.

In the second effort, many theories have been proposed to explain how creativity occurs, from all of them, we would like to highlight The Honing Theory, created by Liane Gabora (GABORA, 2010a), which proposes that a creative idea is the result of an alternating process between associative and analytic modes within the brain. In the associative mode, features from the surrounding environment, sensed by the brain, fire clusters of neurons that were trained to detect those features, which leads to other concepts, with more or less intensity, related to them. This exploration phase is then switched to an analytic phase which chooses, based on a more logical thinking, the ideas which are worth pursuing. This alternation of modes eventually leads to a creative idea. Thus, this thesis main objective is to propose and develop methods to deploy human creativity into an agent's reasoning process. It was divided into three parts.

In the first part, this research envisioned the creation of a computational model of The Honing Theory of creativity, that has never been attempted before, which led us to the formulation of a computational model of it. This model was deployed in a system we called HoningStone to generate creative card combos for a digital collectible card game called HearthStone. The challenge was to generate creative card combos, a set of cards that when played together outpasses their individual powers. This implies in searching a huge space of possible card combos, which makes traditional approaches inefficient. Our results showed that HoningStone could generate combos that are more creative than a greedy randomized algorithm driven by the same creativity metric. An emergent network representation would be more elegant and desirable, since new cards and mechanics are frequently introduced, invalidating the static semantic network and taking hours of human expertise to build a new one.

The second part envisioned to enable an agent to build an emergent solution, not only to build combos but to generate full strategies which at first are just efficient, not creative. In this new scenario, the search space is even larger due to the number of cards and actions combinations and also due to randomness. We then proposed an Adaptive Neural Network for emergent agent development and control using semantic information.

This proposal was deployed on Hearthbot, an autonomous agent we created that plays Hearthstone. Results show that HearthBot achieved on its best version, on average, a win rate of 80% against the MCTS, which is a state-of-the-art algorithm. However, it was not able to beat a handcrafted algorithm based on greedy local decisions.

The last part envisioned to make HearthBot also creative and more efficient in terms of win rate performance, where the Honing Theory was fit within the natural learning process of the Adaptive Resonance Theory. It led us to the proposal and creation of the Honing Adaptive Resonance Process (HARP). It is composed of Adaptive Neural Networks we design to handle creative thinking. Moreover, the agent actions were also not purely random selected when exploring new possibilities, instead, it used the Bayesian surprise to unveil possibilities not previously attempted. This creative Hearthbot agent surpassed the performance of the original Hearthbot and it achieved a win rate performance above 90% against the MCTS and above 60%, in average, when playing against the handcrafted algorithm based on greedy local decisions. In addition, the proposed creative neural model is more efficient in terms of storage space spent and execution time.

1.4 Overview

This section presents an overview of the objectives and proposals to facilitate the reading. This thesis main objective is to propose and develop methods to deploy human creativity into an agent's reasoning system through ANNs. It was divided into three parts that led to the proposal and creation of the HARP. The HARP is the fusion of the Adaptive Resonance Theory (ART), that describes how the human brain store and retrieve memories, and also aspects of The Honing Theory, that depicts how the human brain generates creative ideas. Its main objective is to deploy emergent creative thinking into agents and facilitate them to solve complex problems. It should be noted that the HARP incorporates aspects of a symbolic computational model of The Honing Theory proposed in Chapter 7.

The HARP enables two major creative thinking roles. The first one is the optimization of a strategy that can enable an agent to act in an environment and the second one is the evaluation of the surprise of each individual experience taken by an agent. When combined, those two roles develop optimized strategies that were created due to the influence of the expected surprise. A strategy developed under those roles is categorized as

new and valuable, what is a well-accepted definition of a creative idea. To realize that, the HARP is assembled with three proposed ANNs, an agent reasoning support architecture called FALCON, that bears it to handle the time and a support architecture that deploys The Honing Theory into the ART.

The first ANN, created to deploy the HARP, is called Expectation ART, and it enables an agent to determine the surprise of an observed environment, by learning in its neurons through a Bayesian surprise model. This is crucial to deploy creative thinking according to The Honing Theory since surprise is what lead humans to explore and generate new ideas, that are different and consequently convey a high surprise. A second proposal, called Proximity ANN, is used to represent observed information in a precise way, due to the fact that with a ARAM model, that uses ART operations to code its neurons, can never specialize. The last one is called Unstructured Areas Multi-channel Adaptive Neural Network (UAM-ANN) and it is used to permit an agent to learn and store more information and it was designed to enable faster information retrieval than the ARAM model. Furthermore, it also enables modularity and information sharing among ARAM fields. Since it is hypothesized that a creative system explores a search space better, thus it demands more storage space to store all the extra obtained data, the UAM-ANN is used in this research as a final structure to deploy the creative thinking, through the proposed HARP, into an agent.

The symbolic computational model of The Honing Theory was evaluated to generate combos in a digital game called Hearthstone. Moreover, the HARP was deployed as agents to play Hearthstone. All experimental design, related to the evaluation of both models, is presented in Chapter 11.

1.5 Document structure

The rest of this document is primarily divided into four parts. The first part is composed of Chapter 2, that presents the related works in computational creativity system and agent reasoning with neural networks, and Chapters 3 to 6, that presents all the necessary background to understand some basic concepts relying ART-based ANNs, the FALCON architecture and The Honing Theory, behind this research proposals. The second part, holding the main proposal of this research, is composed by Chapter 7, that presents the computational model of The Honing Theory and the HARP proposals, and Chapter 8,

that presents all proposed ANNs. The third part handles the Hearthstone environment, how to model, represent and code it. All Hearthstone proposed models are presented in Chapter 9 and all agents, HoningStone and HearthBots, are presented in Chapter 10. The fourth part of this document is composed of the results and conclusions chapters, 11 and 12 respectively.

In addition, this document also possesses four appendices. In Appendix A, is discussed and presented a practical example of the ART. On the other hand, in Appendix B, are presented complementary evaluations of the HARP when playing Hearthstone. Moreover, the time complexity and performance analysis for the proposed UAM-ANN, when deployed inside the HARP, is presented. Finally, in Appendix D, the Bayesian surprise behavior for the HARP is presented and discussed.

1.6 Contributions and publications

The main contributions of this research are summarized as follows.

- The first symbolic model of THT (The Honing Theory) based on Knowledge-Based Systems. This contribution was published in (GóES et al., 2016) and enabled three other Master's Thesis in being developed and their results to be published in (JUNIOR et al., 2016), (RAMOS; GOéS, 2016), and (AMORIM et al., 2017).

- The HARP (Honing Adaptive Resonance Process). Composed by the proposed ANNs described below, it is the fusion of the ART (Adaptive Resonance Theory) and THT (The Honing Theory), and it enables an agent to develop creative strategies to solve complex problems.

- The Expectation ART ANN. It enables agents to sense the novelty of a decision when exploring a search space. It incorporates the Bayesian surprise into its dynamics, to hold temporal information considering its experience, to perform advanced cognition and decision making that can lead to novel solutions.

- The Proximity ANN. Published in (SILVA; GóES, 2017), it enabled our agents to model and store information in a compact way, about its environments, with a precise categorization method. It is mainly used to enable the HARP spending less space to store received stimulus from an agent's environment.

- The UAM-ANN. It is a novel ANN able to represent semantic information hierarchically, in a similar fashion to Deep Convolution Neural Networks, and it was primarily designed to handle the amount of information provided by a creative reasoning system.

- Behavioral and non-behavioral feature and action models for the digital collectible card game HearthStone. It is proposed in this research handcrafted and atomic feature representation for ANNs. Moreover, are also proposed ways to represent its reward functions used by the Q-Learning and reactive temporal optimization methods.

- A benchmark evaluation of the Metastone simulator, where it was used in this research to evaluate all deployed agents when learning and playing HearthStone.

- Deployment and creation of HearthStone agents, able to learn and play HearthStone, simulated by Metastone, and used to evaluate all proposals.

Our secondary contributions include a performance and complexity analysis of the proposals, an analysis of the Bayesian surprise behavior for decision making when controlling agents, and practical examples of the ART when deployed in computational systems.

Part I

Related works and background on agent reasoning and computational creativity systems

2 RELATED WORK

This chapter provides an overview of some state of the art approaches related to agent reasoning with emergent structures, such deep neural networks and ART systems; representation of semantic information with ART techniques; and in Computational Creativity for automated systems, agent control and creativity assessment methods. It is divided into three prime sections as follows:

a) **Computational Creativity applied to automated systems**: Discuss automated Computational Creativity Artificial Intelligence systems in domains like fashion, artistic painting, culinary and agent reasoning.

b) **Agent controlling in Digital Collectible Card Games**: Present how agents perform reasoning in the domain of the Digital Collectible Card Games.

c) **Agent controlling with neural networks based on Q-Learning**: State of the art techniques for agent reasoning through neural networks with Q-Learning are presented.

d) **Semantic information representation and memory sharing to enhance agent reasoning**: Presents how information with Adaptive Resonance Theory based systems can be used to constitute semantics in a distributed way.

2.1 Computational Creativity applied to automated systems

In Computational Creativity, an assessment method makes possible machines generate and evaluate creative artifacts (BODEN, 2004). It was realized the importance of a consensual way to evaluate creative thinking since (AMABILE, 1982; PARTRIDGE, 1985; ORTONY; PARTRIDGE, 1987), where machines started to act in a similar way as human beings. At that time, most of the assessment methods were based on psychology, describing it as a mental process that involves surprise, expectancy, and luck (STIENSMEIER-PELSTER; MARTINI; REISENZEIN, 1995). During the development of a consensual method to evaluate creativity, surprise started to be used to create artificial agents as presented by (MACEDO; CARDOSO, 2001; MACEDO; REISENZEIN; CARDOSO, 2004), to model novelty into its behavior or design. Assessment methods evolved as a set of mathematical equations that describe creative artifacts as a combination of surprise,

novelty, and value as shown by (GRACE et al., 2014). To this date, researchers tend to agree that an artifact has to be new and valuable on a particular domain to be considered creative (JUNIOR et al., 2016; BODEN, 2015; COLTON et al., 2015; VELDE et al., 2015).

Creativity evaluation is used in many areas, like artistic painting, scientific, digital games, and even music. Between the related works, the artistic field, the research conducted by (KOWALIW; DORIN; MCCORMACK, 2012) evaluates how creative a painting is for human judges. The proposed system is called EvoEco, and it was deployed as a genetic algorithm. The creativity is measured in the proposal through the Dorin/Korb metric, where it tells that any randomized system has the potential to generate creative artifacts if executed during an infinite amount of time. In contrast, the work realized by (DIPAOLA, 2014) proposes a system called DarwinGaze, and it generates a creative artistic painting through genetic programming. The system is capable of generating random and abstract painting that, besides being abstract, were considered creative by human specialist judges. The proposed solutions differs from the most proposed theories about creative artifact generation as pointed by psychology and neuroanatomy (GABORA, 2010b).

On the field of fashion, the work presented in (KIM; CHO, 2000) uses a genetic algorithm to generate creative fashion designs. The algorithm is driven by a human objective function, where its value is given by human experts and used to distinguish creative from non-creative sets of clothes. In the proposal, the searching space was too small, thus it is reasonable saying that it was not a big challenge searching for creative solutions. Another observation from this work is the fact that using human judges do not seem reasonable for a fully autonomous system since its evaluation is bounded to the response time of experts. Differently, (CHENG; LIU, 2008) uses a semi-autonomous approach to design creative cloth sets, where the system was trained through supervised learning techniques to learn what is considered a creative and non-creative set. A third, more sophisticated approach proposed by (ZHANG; YANG, 2013) to generate creative fashion designs, is more focused on learning how humans evaluate creativity through a radial basis neural network (WENG, 2012b) that is used to guide a genetic algorithm. The creativity function approximation was crafted in order to reduce the need of using human judges, thus turning the solution more autonomous than the previous ones.

On the culinary field, the research proposed by (VARSHNEY et al., 2013) is focused on generating creative culinary recipes. The system uses a novel creativity metric based on the Bayesian surprise. A further extension of is the research conducted by (AMORIM

et al., 2017), where the authors have used a formalized creativity assessment method called Regent Dependant Creativity, proposed by (JUNIOR et al., 2016). The proposed evaluation uses a knowledge-based system in the form of a synergy network together with a Bayesian surprise system, thus being able to calculate the value of a creative idea by adding a synergy relation between concepts, extracted from the proposed synergy network, plus the novelty, calculated through Bayesian surprise, of the generated artifact. The proposal presented by (AMORIM et al., 2017), uses the same principles of computing the Bayesian surprise based on the molecular interaction between flavors for the human taste for already generated recipes, but in addition, it uses the synergy network in order to generate valuable artifacts. The proposed method assumes that creativity can be measured with a single equation, thus simplifying the optimization process. However, much of the work that refers to creativity explains that it is generated not in terms of evaluation, but rather in terms of the process and even the interaction within reality.

With respect to agent controlling, the work proposed by (AGUILAR; PéREZ, 2017) presents an agent controlling scheme that permits it to interact with an environment, where the author had deployed a humanoid agent that is able to interact with objects through its arms. An interesting point about this work is related to the fact the agent incorporates an attention module that guides its will and preference in interaction with certain objects. The agent is controlled by *schemes*, where each one is responsible to hold information on how its actions, performed under an observed environment, influencing its mental state. The authors also propose the idea that an agent controlling system is considered creative if it obeys the criteria of novelty, utility, emergence, adaptation, and motivation. In contrast with other solutions in generating creative artifacts, this proposal seems based on experience, where through the interaction with reality a system can generate creative ideas. This approach is also more similar to the ones adopted to control agents in real time with neural networks, such as the deep learning and FALCON (MIYASHITA et al., 2017; ZHAO et al., 2016; KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017; TAN, 2004; TENG; TAN, 2015), but it uses a symbolic model of reality provided by a specialist that codes actions and environments.

In contrast to (AGUILAR; PéREZ, 2017), the research conducted by (AUGELLO et al., 2017) proposed in using a deep learning network to teach a robot how to dance. The network is trained with several skilled dance moves, where the robot can perform each one of them. All moves are performed in time with the rhythm of the music at each

beat. The robot can perform well at a variety of songs and even in songs never listened before by it. All generated steps during a song are improvisations and consequently can be considered creative when evaluating novelty. In contrast with other creativity assessment metrics, the proposal seems to be able to generate novel solutions, but it fails to represent explicitly what creativity is according to (JUNIOR et al., 2016; BODEN, 2015; COLTON et al., 2015; VELDE et al., 2015).

An interesting work on how robots can behave creatively was developed by (FITZGERALD; GOEL; THOMAZ, 2017). The conducted research points how a robot can behave creatively by deciding what actions to take in order to be considered novel. In order to learn new tasks, the robot can asks for human assistance, where a human teacher can teach the robot new actions to perform. Furthermore, the learned new actions can be used in different environments. The proposed work demonstrates some concern about how creativity is deployed under agents and it also addresses how the embodiment, how the agent perceives and interacts with reality, plays an important role in deploying creative robots. By contrast with other approaches, the proposal seems not to use any neural architecture in order for an agent to reason or learn new stimulus, thus it shows that is feasible to deploy creative agents to perform simple tasks without using emergent learning methods.

In this research, is proposed the HARP, based on Adaptive Neural Networks to develop creative thinking as a mental process that involves surprise, as argued by (AMABILE, 1982; PARTRIDGE, 1985; ORTONY; PARTRIDGE, 1987). The HARP is rather emergent as adopted by (AUGELLO et al., 2017), than symbolic as argued by (STIENSMEIER-PELSTER; MARTINI; REISENZEIN, 1995; MACEDO; CARDOSO, 2001; MACEDO; REISENZEIN; CARDOSO, 2004; GRACE et al., 2014), developed by (KOWALIW; DORIN; MCCORMACK, 2012; DIPAOLA, 2014; KIM; CHO, 2000; CHENG; LIU, 2008), and (ZHANG; YANG, 2013; VARSHNEY et al., 2013; AMORIM et al., 2017; JUNIOR et al., 2016; AGUILAR; PéREZ, 2017; FITZGERALD; GOEL; THOMAZ, 2017). The proposal cares for adaptation and motivation, characteristics of human creativity concerned by (AGUILAR; PéREZ, 2017). Moreover, human expertise is not used on this research to develop creative thinking during learning stages, as used by (KOWALIW; DORIN; MCCORMACK, 2012; DIPAOLA, 2014; KIM; CHO, 2000; CHENG; LIU, 2008; ZHANG; YANG, 2013) and (FITZGERALD; GOEL; THOMAZ, 2017).

Luck is a characteristic of human creativity addressed by (STIENSMEIER-PELSTER;

MARTINI; REISENZEIN, 1995) and (MACEDO; CARDOSO, 2001; MACEDO; REISENZEIN; CARDOSO, 2004). In this research, it is not used as a basis in which creative thinking can emerge since these research proposals assume a controlled exploration and self-development scheme. On the other hand, expectancy is deployed within an Adaptive Neural Network's prediction routines since it is important to predict what could possibly help an agent in solving problems.

Surprise and novelty seems to be reasonable ways to measure how new an idea is, thus being able to calculate a creative level of a generated artifact or idea as argued by (MACEDO; CARDOSO, 2001; MACEDO; REISENZEIN; CARDOSO, 2004; GRACE et al., 2014) and adopted by (KOWALIW; DORIN; MCCORMACK, 2012; DIPAOLA, 2014; KIM; CHO, 2000; CHENG; LIU, 2008; ZHANG; YANG, 2013). But in this research, it is used only the concept of novelty as adopted by (VARSHNEY et al., 2013; AMORIM et al., 2017; JUNIOR et al., 2016), since it can lead to equal results. A utility value is also utilized, in this research, to guide the creative agent reasoning process from the HARP, as accomplished by most of the observed researchers from the literature since it is a common variable that helps to guide an optimization process and it is also a part of the creative thinking definition.

Novelty is developed by the solution in an emergent way by a proposed Adaptive Neural Network called Expectation ART, differently from (VARSHNEY et al., 2013; AMORIM et al., 2017; JUNIOR et al., 2016) that calculate the Bayesian surprise with the help of a knowledge database. Furthermore, the utility value is also utilized to develop creative strategies, and it is optimized by a learning technique called Q-Learning. This technique allows the agent to realize sequence of events and the existence of time when generating creative strategies.

The proposal does not rely on randomness or evolutionary approaches as developed by (KIM; CHO, 2000; CHENG; LIU, 2008; WENG, 2012b; DIPAOLA, 2014). But uses neural networks to allow an agent to develop by its own. However, differently, from (WENG, 2012b; AUGELLO et al., 2017) that use supervised learning, it uses proposed neural networks to resemble The Honing Theory of Creativity and The Adaptive Resonance Theory to deploy unsupervised learning and reinforcement learning. This nature is what yields the solution to develop on its own and also adapt to a distinct set of situations according to its needs and in dynamic changes in an environment, what is not accomplished by other proposals.

As a final remark, with respect to Computational Creativity, the proposed HARP is an emergent neural system, differently from (KOWALIW; DORIN; MCCORMACK, 2012; FITZGERALD; GOEL; THOMAZ, 2017; VARSHNEY et al., 2013; AMORIM et al., 2017; JUNIOR et al., 2016; AGUILAR; PéREZ, 2017) that are symbolic. This is due to the fact that the HARP is an Adaptive Neural Network neural system that simulates a POMDP used to allow an agent to model its reality and develop creative thinking. Since the proposal simulates a POMDP, one of its prime objectives is to optimize and determine an optimal path that resembles a strategy. The proposed solution confirms a creative idea is a POMDP path or strategy, where its optimization is influenced by novel decisions, when the agent is performing in its reality, triggered by implemented aspects of The Honing Theory. As observed from the analysis of the Computational Creativity related works presented in this document, this approach was never attempted before and it is in conformity with two theories that describe how the human brain works.

2.2 Agent controlling in Digital Collectible Card Games

Digital card games are a fertile territory for computational intelligence since they have both incomplete information about the opponents cards and randomness in card drawing (WARD; COWLING, 2009b; BURSZTEIN, How to appraise hearthstone card values). Ward (WARD; COWLING, 2009b) proposes the use of MCTS to select cards to play on each turn in Magic: The Gathering. Like Hearthstone, Magic: The Gathering has incomplete information since a player does not know the opponents cards, making MCTS a suitable algorithm as it evaluates possible moves on each turn instead of all the moves. Thus, the MCTS approach is tested in simulations against a rule-based bot designed by an expert.

In (WARD; COWLING, 2009c), the authors use a genetic algorithm to generate a set of cards that balances the card game Dominion. Based on three fitness functions, the results showed that there are specific cards that govern game balance independently of player skill and behavior. In contrast, (SEPHTON et al., 2014) proposes a move pruning heuristic for MCTS in the card game Lord of War. They experimented a single hard pruning heuristic, multiple heuristics and state-extrapolation, in which results show that the same heuristic could be applied to other domains. For partially observable games with simultaneous actions, Teytaud (TEYTAUD; FLORY, 2011) proposed an extension to UCT

(Upper Confidence Tree) and applied to a card game called Urban-Rivals. With respect to card evaluation, (BURSZTEIN, How to appraise hearthstone card values) proposes a simple linear model to evaluate Hearthstone. It shows which cards are undervalued so players can build more efficient decks and combos using those cards. While this model works for basic cards, it breaks for more recent cards and requires careful modeling of each card with specific effects, thus resulting in a handcrafted model.

When dealing with digital collectible card games, it seems that the MCTS approach is quite popular, due to its fast and easy to implement algorithm. It can be also observed that MCTS approaches, even working together with deep learning methods (SILVER et al., 2016), are used often to control agents in an environment with a higher search space for selecting actions, like board and card games, because it is fast to handle action selection by doing randomized sampling at a tree structure used to represent game states. From a digital collectible card games perspective, there is a gap on the literature with respect to autonomous agents applied to *Hearthstone*, where a great part of the literature deals with Magic: The Gathering due to the fact that it is a more popular, older game.

Since the proposals from this research are emergent, an approach based on the Monte Carlo heuristic adopted by (WARD; COWLING, 2009b) was not used. Instead, the agent do not explore a search tree but rather optimize a simulated POMDP path that is learned and created during the game. Furthermore, the proposals assume that the learned POMDP can suffer modifications if necessary, thus giving more room for it to adapt into different situations.

The first contribution of this research, the symbolic computational model of The Honing Theory, was also used to generate artifacts, as also accomplished by (WARD; COWLING, 2009c), however, the primary contribution is used to generate broad strategies that effectively permit it to play Hearthstone.

As accomplished by (SEPHTON et al., 2014), the proposals also use a pruning method to discard places that can not contribute to beneficial results. However, in this research, the pruning method is implicit in the POMDP path optimization technique called Q-Learning. This technique creating a trail in which the agent should follow to win the game. To guide the Q-Learning optimization, a card evaluation scheme was also utilized, and it was based on the evaluation method presented at a HearthStone simulator called Metastone. The proposed evaluation methods seem to be more robust in terms of variables considered and in a working range to evaluate cards than the methods used

by (BURSZTEIN, How to appraise hearthstone card values) to generate decks, since it considers the Q-Learning's behavior and numeric interval.

Differently, from (SILVER et al., 2016), this research proposal does not rely on deep learning approaches to learn how to compete in the game. It can be noted that the presented related work used two networks to evaluate the game and generate viable strategies, however, it was based on a dense Graphics Processing Unit cluster that demanded vast amounts of energy. On the other hand, this research proposes that two Adaptive Neural Networks, designed to develop creative thinking, can play a digital game with less computational effort and also able to obtain reasonable performance results against human-like agents. Furthermore, the proposal enables an agent to learn faster than deep learning approaches, what can be crucial to perform in time if acting in a time-constrained environment.

2.3 Agent controlling with neural networks based on Q-Learning

The Q-Learning technique has been widely used to control and optimize agent behavior through time, thus dealing with the delayed reward problem (TAN et al., 2008). As introduced by (MNIH et al., 2015), deep learning together with Q-Learning can be a useful tool when creating a system that is able to play a game and behave like a human player. This kind of system can develop strategies over time that considers how an agent should behave according to what happened or what will happen. As shown in the subsequent paragraphs, deep learning approaches with Q-learning are being widely used, since they provide flexibility giving full autonomy to the learning system. However, it is important to note that those kinds of networks are bounded to its static structure of layers and neurons, thus obligating the system to learn on a structure defined a priori. In contrast with deep learning approaches, the class of networks called temporal FALCON (TAN et al., 2008), seems also suitable to play games by simulating a POMDP. However, it lacks the ability to approximate functions, because of its fast and stable learning mechanism, thus not being as suitable as deep learning approaches for function approximation.

One of the most famous attempts developing an adaptive agent, where the authors AlphaGo, is an artificial intelligence that is able to play a game called GO (SILVER et al., 2016). The proposal is based on a deep learning neural network that analyses the game patterns, that evaluates states through a policy network, performs Monte Carlo rollouts,

and a value network, that is able to tell if a state looks like a winning state or not. The solution the two evaluations into a MCTS tree search process in order to select an action that maximizes the agent probability of victory. The proposal was able to win against one of the most virtuous players of the Go game on the world by a score of 5 x 0 in favor of the bot. This research shows that deep learning approaches are capable of controlling agents, however, though requiring a higher computational cost. The proposed technique was deployed into a 1202 Central Processing Unit cluster that has 176 Graphics Processing Unit. It can be observed the proposal relies on visual pattern recognition and searching substantial amounts of states through a massive parallel power, thus its high accuracy when performing in a computer cannot be ensured.

One of the most recent works on Q-Deep Learning is presented in (MIYASHITA et al., 2017), where the author proposes a framework able to incorporate reinforcement learning and supervised learning, called separated network shared network models. They have used two multi-layered networks with three convolution layers and two fully connected layers to deploy reinforcement, supervised and mixed learning. The results were applied in a game where the main objective of a player is to obtain an apple that pops up at a random position in a grid world. This work is directly related to *Computer Vision* since what the agent is codding from the game is based on what it is perceiving. This method seems suitable in recognizing patterns, however, one of most concerning problems is related to the fact that multi-layered networks have a higher learning time, thus their application was trained with 10^5 frames that contain human moves captured in logs. Another observation that can be drawn from this work is the fact that it involves two types of networks, where the supervised type comprehends information obtained from a specialist and a reinforcement type learns how to play with Q-learning. Nevertheless, it seems that the primary object of the proposed work is to train agents that behave more human likely.

Differently, from (MIYASHITA et al., 2017), the work proposed by (KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017) presents a general bot that works with a deep learning Q-Network in order to play any kind of game. This network is effective to relate visual patterns with convolution layers and a dense layer, where the recognized patterns are used to train a Q-Learning algorithm. The proposal seems robust enough in learning to compete in various games with a unified structure. Furthermore, the trained agent can play a series of games by perceiving a broad environment and performing an image pattern

recognition technique. However, when using a deep neural network, an agent can not handle learning and performing routines at real time, thus it is desirable that the network's structure can adapt, into flimsier structures, according to the game domain. Furthermore, the authors also show that knowledge transfer can help when performing in real time by using previously acquired knowledge into unknown environments.

Another work that had used the Q-learning technique is the research conducted by (ZHAO et al., 2016), where the author has successfully applied the Q-learning technique with a temporal error calculation rule called *State Action Reward State Action*. As in (MIYASHITA et al., 2017) and (KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017), the proposed solution uses pattern recognition from visual patterns from games. It can be observed from this work that, the author had operated a Deep Q-Learning network, that handles the Q-learning learning equations inside its loss function when adjusting neuron weights from a bunch of convolution layers. The Q-learning is optimized in this work by a gradient-descendant method. It can be observed that the results show that the agent's behavior converge into a local optimum but it also seems unstable around its convergence point. That behavior could be related to the fact that the network's learning method is sensitive and slow, due to convergence problems related to the model used to represent the information obtained from the game and even the optimization method used.

One of the most related work was the proposal of an agent controlling architecture by (TAN, 2004), where an adaptive neural network, based on fuzzy ART operations, with a FALCON is used to control agents in real time. This FALCON model encompasses a reactive model based on sense, thinking and acting, thus an agent was able to navigate through a hypothetical minefield. A further extension to the FALCON model was the temporal FALCON (TAN et al., 2008), where temporal difference methods were incorporated to perform reasoning considering time. In a recent effort to control agents in real time, (TAN, 2004) proposes combinatory operations to be used instead of fuzzy ART, thus resulting in a more precise mechanism to control the agent in a game called *Unreal Tournament*. Both proposals seem suitable when controlling agents in real time, however, they do need a generalized input, that can represent invariant information about an environment, thus they are not suitable in dealing with precise semantic information.

One of the advantages when using the proposal from (TAN et al., 2008), is the fact that the system is stable and encompasses a fast learning fuzzy technique. By using this fast learning and stable method the proposed system was able to compete in a game in

real time without having issues to represent information neither in storing it inside the network's neurons. This kind of approach differs from deep learning ones with respect to function approximation, where the deep learning methods try to map the entire game environment as a single equation coded in a distributed way inside its neuron layers. On the other hand, the approach proposed by (TAN et al., 2008) tries to store every possible pattern from the game as an adaptive cluster of neurons, that can grow or shrink, to constitute unique stages of a game. There is a payoff when employing this method, where if using it an agent will no longer be able to precisely match patterns for function approximation.

In contrast to (TAN et al., 2008), the work proposed by (TENG; TAN, 2015) uses a double Q-learning approach in order for an agent to play the minefield game. The proposed technique seems to converge better and faster than conventional temporal FALCON models, it is important to note that the proposed minefield problem is relatively easy to solve and conventionally fits the ARAM fuzzy input coding. Furthermore, the explored minefield, in (TAN, 2004) and (TENG; TAN, 2015), the agent navigate through a 10×10 size minefield grid with 10 mines. It seems reasonable to assert that if the environment or the quantity of the mines changes over time, the performance of the agent should not be as good as presented by the authors. That stems from the fact that the search space for the solution will be bigger and harder to explore.

According to most part of the work presented at the state of the art, the Q-Learning technique seems to be performing consistently through various domains. Deep learning approaches are quite a popular and seem to be used often to recognize patterns on images and thus an agent was able to represent a virtual Q-Learning table, and not to form a higher cognition system that deals with semantic information. The main problem when using deep learning approaches is the slow learning rate that those networks tend to have, thus increasing the learning time for an agent when interacting with an environment. Another problem is that deep learning approaches tend to mimic image filtering processes through its layers, thus most applications tend to have a larger network size that can compromise the ability to be used by low-end computers. The last and one of the major problems is that the network learned stimulus is unstable, sensitive to environmental changes since they try to simulate function coefficients among their structure composed of neurons. Furthermore, it seems that fuzzy approaches, with networks from the ART family, can do a better job when doing fast and stable categorization. Especially, the

ARAM network can map a virtual POMDP with the FALCON architecture, thus an agent was capable of understanding its environment to take decisions. Some extensions for the ARAM network had also used Q-learning efficiently, but it is important to note that all the presented work that use this class of network deal with simpler problems where image classification and feature extraction are not dealt by the system.

To deploy the HARP, the primal contribution of this research, it is proposed three Adaptive Neural Networks. The solution is also based on the Q-Learning method as the deep learning approaches (MNIH et al., 2015; SILVER et al., 2016; KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017; MIYASHITA et al., 2017; ZHAO et al., 2016) and the ones based on Adaptive Neural Network (TAN, 2004; TAN et al., 2008; TENG; TAN, 2015). Despite of that, the proposal does not use a double Q-Learning technique, as encompassed by (TENG; TAN, 2015) and do not use a Gradient Descent optimization technique, as done by (ZHAO et al., 2016), since the primal objective is to deploy a creative thinking system and not to enhance the Q-Learning method.

Since the solution is based on Adaptive Neural Networks, as also adopted by (TAN, 2004; TENG; TAN, 2015), it primarily permits the learning system to adjust its size according to its needs, to encompass more knowledge or change a previously learned one, permits fast learning, enable real-time learning, and it is not dependent on a Graphics Processing Unit cluster. Furthermore, the system also encompasses reinforcement learning and non-supervised learning, moreover, it also supports supervised learning. Finally, it does not encompass visual pattern recognition techniques, as adopted by (MNIH et al., 2015; SILVER et al., 2016; KUNANUSONT; LUCAS; PéREZ-LIéBANA, 2017; MIYASHITA et al., 2017; ZHAO et al., 2016), since all data from an observed environment is collected directly from a simulator through a symbolic data model.

One of the proposed ANNs, called Expectation ART, is used to learn and calculate the Bayesian surprise, since to this date there are not neural networks capable of achieving this for real-time creative thinking. This network is based on a fuzzy ARAM, as used by (TAN, 2004; TAN et al., 2008; TENG; TAN, 2015), however, in its internal structure, memory neurons are used to store information about POMDP states. Those memory neurons can store any kind of data and are used to store information about which action is most often taken by an agent when performing, thus allowing to take decisions upon that fact. Despite being able to act as a neural network alone, the most important contributions of this proposal are its neuron learning and storage models, based on the Bayesian surprise,

not used by any of the related works, to recover temporal information about states and to know the surprise level of a perceived environment.

In addition to the Expectation ART, it is also proposed a proximity-based ANN. This network was proposed to permit precise categorization, learn and retrieve knowledge as close as possible to what was sensed, to deal with situations where semantic information cannot be generalized during learning. It is also based on the fuzzy ARAM, as used by (TAN, 2004; TAN et al., 2008; TENG; TAN, 2015), however, its major difference is that it uses a Euclidean distance based metric for learning and it was deployed into a Graphics Processing Unit to verify how many neurons need to be stored in order to simulate a POMDP with minimum generalization. If seems from the proposed HARP perspective; this network is used alongside with the Expectation ART to deploy creative thinking.

The last proposed ANN was used to permit the HARP to perform in real-time, handle more information than a typical ARAM based system, and it is presented at the subsequent section.

2.4 Semantic information representation and memory sharing to enhance agent reasoning

As argued by (TAN, 2004), the ARAM model is a set of multiple ART maps to allow information sharing and a multidimensional field mapping to represent an environment. The main problem is that the network growth is exponential according to the number of fields used to represent its knowledge. Furthermore, it demands a high computational cost to perform in real-time. There is one major work that used multiple ARAMs to represent knowledge, trying to avoid that overgrowth and to enable memory sharing, developed by (WANG; TAN; TEOW, 2017), where they are used to represent memory through a centralized managing structure. The main problem is that (WANG; TAN; TEOW, 2017) work with several ARAM modules that do not share their fields and consequently falls in the same problem as the single ARAM model. In order to tackle both problems, it is proposed the UAM-ANN. The main difference from the prior related works is due to the fact that the UAM-ANN is assembled as a single structure, as a Convolutional Neural Network, where each level has a different semantic meaning. Furthermore, the UAM-ANN permits fast information recovery and to store large amounts of data due to a tree structure layout. Differently, from (WANG; TAN; TEOW, 2017), the UAM-ANN is

mainly designed to serve as a replacement for each individual ARAM, faster and to handle more information.

Differently, from all related works documented in this research, information is represented by meaning levels to deploy a FALCON architecture that can retrieve information about the simulated POMDP in a faster fashion and also enable to store information about more states. Each level of the UAM-ANN can represent different kinds of neurons, what is similar in deploying ARAM fields with different kinds of neuron coding techniques. Furthermore, each level can also predict by its own, what is also different from the original ARAM structure. Since it is hypothesized that a creative system explores a search space better, thus it demands more storage space to store all the extra obtained data, the UAM-ANN is used in this research as a final structure to deploy the creative thinking, through the proposed HARP, into an agent.

3 AGENT REASONING

This chapter portrays a background on agent reasoning process since it is crucial to understand how the concept of time is simulated by computer programs do deploy a reasoning system. It shows how an agent sees and interacts with a virtual world that simulates time through the concept of a *Time Flow*, concept described during the 70's by (SZYGENDA; HEMMING; HEMPHILL, 1971) as a way to represent time and sequence of events, simulated by a POMDP.

What enables an agent to perform actions on an environment are processes that control its behavior though structural layers (BROOKS, 1986; GULDNER; UTKIN; BAUER, 1995). Those structural layers are typically assembled with an *Actuation* layer, responsible for all its motors, devices and sensors that directly interact with an external world, a *Hardware* layer, that holds all the necessary hardware to control its behavior through programs, and an abstract *Reasoning* layer, that permits it to reason about what it will do, what it can do, what will be more profitable for it in terms of completing objectives. This layered system is assembled, in each of its layers, as a hybrid control system between reactive and deliberative solutions (ABDELLARIF et al., 2012; BRUNETE et al., 2012).

Considering the reasoning layer, the main problem is the lack of a mechanism that empowers agents with reasoning to decide what is good and what is bad in any environment e. For example, in a set of n actions that an agent can perform upon the presentation of an environment e, it needs to define which one will result in the most profitable reward leading to a new environment e_{new} that favors the agent's existence. A system that solves this problem can be created in two ways: *Emergent*; and *Symbolic*. Symbolic solutions are created by a specialist and they are assembled by hand, where a specialist defines for the agent what it will be considered good and bad. Moreover, emergent solutions are self-deployed and created by the agent itself with the help of learning techniques.

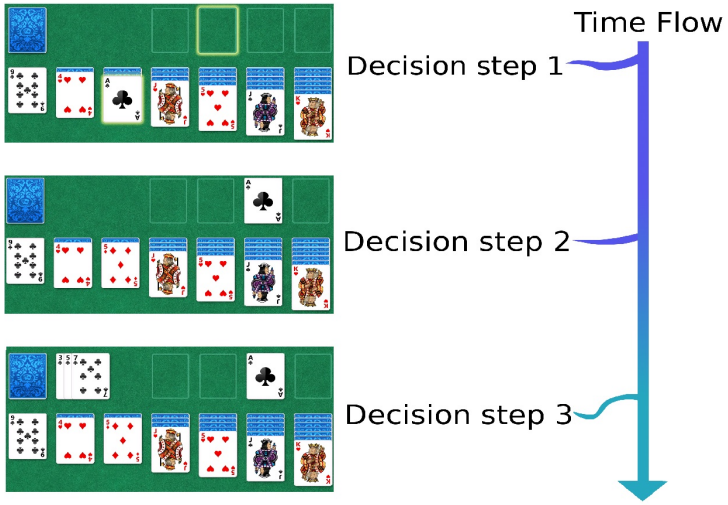
Emergent solutions are created to this date with the help of neural networks such as the *Temporal Context Machine*, ART, *Deep Learning Artificial Neural Networks* (WENG; LUCIW; ZHANG, 2013; TAN et al., 2008; WANG; TAN, 2015; WENG, 2012b). One of the most concerning problems of using a neural network with reinforcement, unsupervised and supervised learning, is that the system will reason exclusively about what it is trained to, not adapting to different situations. To tackle this problem, a cognition architecture, such as the FALCON (TAN, 2004), is used to enhance the neural network's abilities. With

this architecture, the neural network stops being a data structure capable of only mapping function values and can be used to grasp the existence of a sequence of events, semantics in the actions performed by the agent and also the existence of time.

3.1 Digital games

When analyzing the behavior of an environment in the real world it can be realized that it has much in common with digital games. For example, a player can feel the sensation of being inside a universe with the notion of time and space. In fact, what happens inside of a game that simulates the concept of time, and all the interactions that happen inside of it occur within a predetermined space. Considering the simplified structure of games in comparison to reality, in this research, the notion of agent controlling is explored with the help of games and related to computer-simulated programs. For instance, the game Solitaire, depicted in Figure 2, is an example of a game that simulates the concept of time, where each move performed by a player is taken in discrete time moments referenced in this research as *Decision Steps*. The main structure where each decision steps interacts is called *Time Flow* and it is a simulation of time (HENRIKSEN et al., 1986), where it defines how the concept of time should be represented in computer programs.

Figure 2 – Time flow in a Solitaire game with discrete decision steps



Source: By the Author.

A *Time Flow* inside a game can be manipulated at will, but from a player's perspective it flows typically towards future. It is used to describe changes in position and state of objects in games. Considering the Solitaire example, time helps in determine the

position of cards, the player score, the available cards to buy, what cards players can move and interact with at different moments. To identify a determined position at the *Time Flow* it is necessary a snapshot of it containing every information about the game that has not suffered any change on a measured interval. This information can be obtained from a window or a singular point at a *Time Flow* and it is called as State and is referenced on this research as TS.

States contain all the information that is used to represent a game. For instance, what players can see when playing games is the information inside a TS_α at a moment α on a *Time Flow*. Time is perceived in games by observing a sequence of States reached by decision steps, where each step has the capacity in changing information inside TS_α leading to a new $TS_{\alpha+1}$. What mainly differs game types, between real time and discrete time, and consequently how players interact with them, is how often States are obtained from a *Time Flow* and used by a player. This model of representing time is widely used to simulate not only games but also computer programs in general (GOLDSMAN; NANCE; WILSON, 2010).

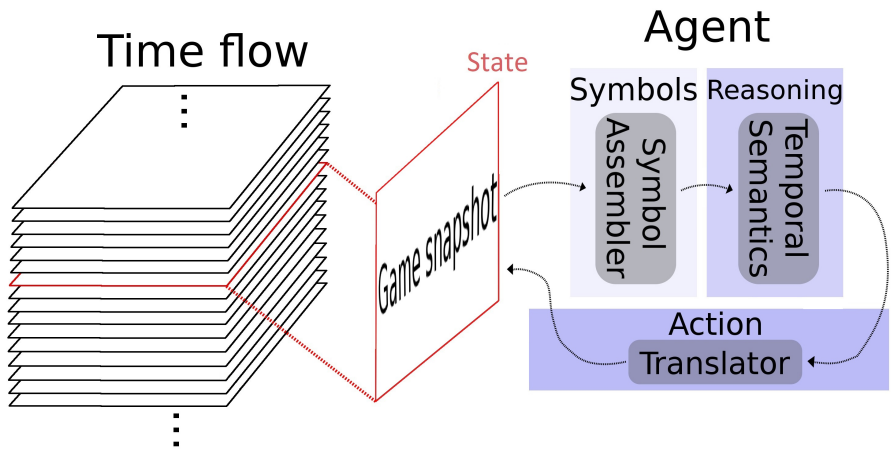
In reality, time is a measurement of the difference from what is old to what is new and this measurement is what creates on humans the illusion of a *Time Flow* and change towards future. Analogous to reality, the ΔTS change on information from a State to another is what gives the illusion of time in a game. It is important to recognize what caused any change on information, because it will permit a player to interact with those sources and maybe win a game.

To establish what can cause a change on information inside a State, they can be organized as a set TS_{super}. This set, can be handled by a computer, and it is responsible to store all States that occurred or will occur on a game in its *Time Flow*. The set TS_{super} is then composed by $\{TS_{start}, ..., TS_i, ..., TS_{end}\}$, where TS_{start} is the initial State at the start of the game, TS_{end} being the final State at the end of the game and TS_i being an intermediate State between the start and end. There can be two kinds of changes in information on a *Time Flow* inside a game: 1) *Known Changes*; and 2) *Unknown Changes*. The *Known Changes* lead to TS_k where k is an arbitrary previously known *States*, that already happened and are occurring again. On the other hand, the *Unknown Changes* lead to unknown $TS_?$, that never happened before, where the symbol '?' is used to represent unknown states of a *Time Flow*.

3.2 Reasoning process

The process of controlling an agent is what permits it to sense the environment, itself, and reason about what *actions*, set of *events*, it should prioritize to be performed when interacting within a *Time Flow* through its *Decision Steps*. As shown by the example of the agent controlling depicted in Figure 3, reasoning occurs as the last stage of a controlling process. This control process starts by observing an external environment and extracting meaningful data from it through the help of techniques from *Computer Vision*, *Signal Processing* and related areas, that information composes a game snapshot or a State. All that data is organized into symbols, sets of meaningful data, representing information about an environment. All assembled information is sent into a reasoning architecture that uses algorithms and heuristics to decide what action an agent should prioritize. The selected action passes through a translator to be performed in an observed environment. This reasoning process is referenced in this research as a *cognition*.

Figure 3 – Agent control process from data acquisition to decision making.



Source: By the Author.

The reasoning process starts after obtaining a State from a *Time Flow*, showed in Figure 3, and it is desirable that it ends before the acquisition of a new State. This avoids losing or delaying information about what is happening inside of a game. Furthermore, an agent can also enter an idle mode, automatic pilot, if the reasoning for a State is taking too much time to finish, thus it will be able to perform default actions affecting less its performance. The frequency of States acquisition defines the genre of a game, from real-time, acquired as faster as a computer can, to discrete time, acquired at specific

moments defined by a game's rules. It is common in associating the complexity of a game, how difficult it is to solve or understand, by its nature, continuous or discrete. However, the amount of captured States is the defining factor for how often a game snapshot is perceived.

3.2.1 Reasoning through temporal actions

Before any reasoning could be achieved, the agent needs to establish what *actions* are important, good or bad, in order to define what its algorithms and heuristics should seek for. That is a problem since there is no standard way to define what *actions* are good or not, thus turning most of the existent solutions in agent reasoning tied in predefined ways to calculate the goodness of *actions*. In order to tackle this problem, in this research, the agent reasoning is guided by experimentation, where if an agent does not know the goodness of an *action*, then it should try performing it in order to learn. The assumed scheme of control is inspired by an agent control architecture called FALCON presented in Section 6. This architecture enables an agent to define its goals by experimentation and can encompass reinforcement, supervised and unsupervised learning.

The goodness of an *action* is stored in the form of a reward that is represented by a function value in order for an agent to recognize exactly which *actions* to prioritize. The primary problem is the lack of a structure to store the reward of each possible *action* that can happen on each State. *Actions* rewards are also influenced by others that already happened and will happen in a *Time Flow*, thus causing a temporal dependence between *actions* when reasoning about which one is better at an observed State. It is essential to note that, rewards obtained from performing actions, can be influenced by other components from the game that can be guided by randomness, thus possibly leading an agent into uncertain States and turning harder the task of defining a general set of *events* that it needs to initiate.

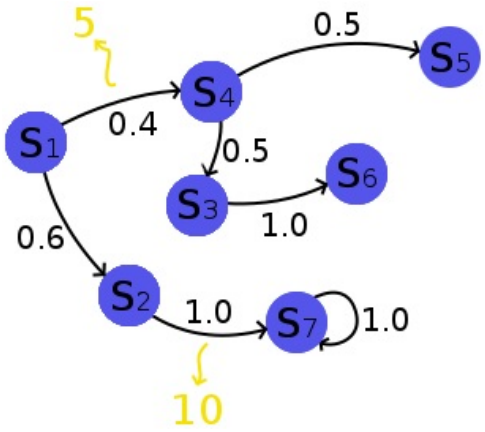
3.2.2 Partially observable Markov Decision Process

In order to model rewards from *actions*, as addressed in Section 3.2.1, this research uses a system based on a POMDP (SONDIK, 1978). It is a numerical probabilistic model that enhances the *Time Flow* representation by associating *actions*, that can be performed in states, with rewards. It is described as a tuple, ordered list of elements, of the form

$(S, A, T, R, \Omega, O, \gamma)$, where S is a set of states, A a set of actions, T a set of probabilities that describe the chance to transitioning between states in S, $R : S \times A- > \Re$ is a reward function used to describe rewards from performing actions, Ω is a set of observations, O being a conditional observation probability, and γ being a discount factor.

For instance, the example depicted in Figure 4 is a game that have been through 7 States, where each *event*, represented by black arrows, enables transitioning between States. Each transition also has a value associated, represented by black numbers, that indicates the probability of going from a State to another by performing its associated *event*. The yellow arrows, on the other hand, indicate the reward of an *event* performed in a State. For example, if an agent is reasoning in state s_1 it can perform two *actions* that can lead to state s_4, receiving a reward of 5, and s_2 with a probability of 0.4 and 0.6, respectively. It is important to note that it is not mandatory that every transition be associated with a reward since it is unguaranteed that every *event* will lead to outcomes.

Figure 4 – Simplified Partially Observable Markov Decision Process example with 7 states in S as blue circles and 2 reward outcomes in yellow.



Source: By the Author.

From a POMDP perspective, the main objective of a reasoning process is to maximize the overall reward of an agent during a *Time Flow* by deciding what *actions* are most important to be performed. In a POMDP, the overall reward of an agent's performance is equal to the sum of all the intermediary ones obtained through its *Decision Steps* over time by using Equation 1, where t is the present *Decision Step*, γ being a scale factor to discount the reward, and r_t being the reward received by performing an *action*. The γ variable is powered by t giving more weight to outcomes that are closer to the end

of a *Time Flow*.

$$E = \sum_{t} \gamma^t r_t \tag{1}$$

Transitions in a POMDP can occur with a probability described as $T(s'|s, a)$, where s is the current state, what the agent is seeing at the present, a being the *action* performed by the agent, and s' being the new state obtained after performing a. This model is coded probabilistically because it can be unguaranteed that the new state s' will be achieved. A POMDP also permits to predict a reward of a future state s' by doing an observation $O(o|s', a)$ inside Ω, where $o \in \Omega$ is the observed future reward from an environment s'.

3.2.3 Strategy as a POMDP path

The reasoning process generates a set of *actions* that an agent needs to perform in order to maximize its overall reward over time, thus generating a path in a POMDP. This path is composed of all actions and states that were performed and the agent went through, respectively, and it is referenced in this research as a strategy. Behaviors of an agent are accomplished by changing or adapting its strategy over time. The main problem when dealing with the generation of strategies with a POMDP, to play a game, is that they are not trustful since performing *actions* can not guarantee reaching desired states. Consequently, every strategy that an agent can generate, through reasoning, represents a blurry possibility from a *Time Flow*.

3.2.4 Observability and POMDP limitations

The observability of a *Time Flow* and a POMDP can be full or partial, where the full observability means that every variable, *event*, logical rules from a *Time Flow*, transition probabilities, rewards, states, are known, otherwise, it is partial. A full observability enables an agent to know everything about the game, thus facilitating its reasoning by alleviating the necessity in dealing with unknown information. On the other hand, when dealing with a partial observability, the strategies that are generated by it are even more blurry and can result in higher statistical fluctuations.

A *Time Flow* can have any size, what implies in an infinite number of states to represent in a POMDP, thus generating a high demand in processing power to run a reasoning process. Furthermore, it is also difficult to store an infinite amount of possible

state by hand, where the most used solutions involve state contractions, fusing states to simplify the POMDP, in order reduce its complexity in terms of size. A major problem when dealing with a POMDP to control an agent, is the lack of a mechanism, unbounded from a labeling system, that creates new states if needed, that stems from the fact that it is impossible to label an infinite number of states.

In order to deal with reward estimation, optimization, and also state labeling and creation for adaptive learning, in this research the POMDP is simulated with a proposed Adaptive Neural Network inspired on an ARAM. The ARAM neural network is based on a theory called ART, where it specifies how the human brain stores and retrieves information in a decentralized way to achieve not only ways to evoke memory but also advanced *cognition*. With its mechanism, the proposed solution is capable to simulate each POMDP state inside neurons.

4 THE ADAPTIVE RESONANCE THEORY

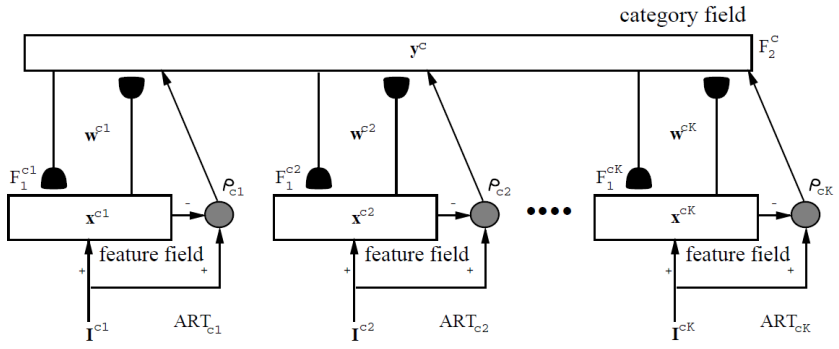
This chapter presents a background on the ART computational model and a fuzzy Adaptive Neural Network capable of implementing it, the necessary tools to deploy the HARP proposal, and also the theory behind all the proposed ANN to handle creative thinking, memory representation, information retrieval, and prediction.

Adaptive neural networks, as the fuzzy Adaptive Resonance Associative Map ARAM and fuzzy ARTMAP, are used in a variety of applications, ranging from computer vision to agent controlling in games (CARPENTER; GROSSBERG; REYNOLDS, 1991; WANG; TAN, 2015). Those networks enable an agent to adapt into various scenarios according to the its needs and are well established computational models of The Adaptive Resonance Theory. This behavior can be useful if the environment that interacts with the agent is unknown or partially unknown. They are based on more than two ART systems that communicate with each other to form an associative memory (TAN et al., 2008; TAN, 2004; TAN, 1995), mapping an environment. Considering that the ARTMAP was introduced first than the ARAM, in the rest of this document, an ART inspired neural network, that uses more than one ART system, is referenced as a multi-channel ANN (Adaptive Neural Network).

In its general form, depicted in Figure 5, the multi-channel ANN has two layers, F_1 and F_2. The F_1 layer is called feature layer, where features are extracted from an environment and arranged there in a coherent way inside the feature fields. Each feature field inside F_1 is represented by F_1^{ci}, where $i \in [1, k]$, being k the maximum number of fields. In contrast, the F_2 layer, called category layer, acts as a multidimensional pattern holder where it has the function of linking all the F_1^{ci} feature fields in F_2 with neurons. Each F_1^{ci} also has an input field $p_1^{ci} \in I$ and an activity field $x_1^{ci} \in X$, where I holds all input fields and X holds all activities from the feature layer. The input field p_1^{ci} receives raw stimulus signals directly from the environment so they can be transformed into a feature vector and transferred to the corresponding activity field x_1^{ci} for usage.

With respect to the F_2 category layer, each neuron j is coded as a set of weights $W_j \in W^{all}, j = 1, ..., t$, where $W_j = \{w_j^{c1}, ..., w_j^{ci}, ..., w_j^{ck}\}$ and W^{all} as the set of all neurons. The variable t is used to identify the neuron limit of the network, $i \in [1, k]$ the link to the field ci inside F_1 and k the total number of feature fields. An input vector is coded as $p_1^{ci} \in I = \{p_1, ..., p_n\}$ and an activity vector as $x^{ci} \in X = \{x_1, ..., x_n\}$ where n is the total

Figure 5 – Adaptive Resonance Associative Map architecture.



Source: By (TAN, 2004)

number of features coded by the field ci. All those fields also need to use the complement coding to prevent the code proliferation problem (CARPENTER; GROSSBERG; ROSEN, 1991; TAN, 1995; TAN, 1992; TAN, 2004). The complement prevents the network rapid growth but its also provides an overhead of data equals to the size of each field. The full mechanism of the multi-channel ANN can be observed in the literature typically described as this five step routine: 1) activation; 2) inhibition; 3) readout; 4) resonance; and 5) learning. This five step routine is presented in details on section 4.1.

4.1 Categorization mechanism

The network has two types of neurons, committed and non-committed. The committed neurons are used to represent already learned stimulus and the non-committed neurons represent an empty or available space for learning. At its initialization, the network has one non-committed neuron that can be used for learning. The five step routine of the multi-channel ANN permits the cognitive code matching, learning and prediction given an observed environment and it is described as:

1) Activation: Given the presentation of the activity vectors x^{ci} from all the feature fields in F_1 to the system, it computes $t_j \in T$ described by Equation 2. T will hold the temperature or similarity between all $w_j^{ci} \in W_j$ and all the presented $x^{ci} \in X$ in relation to W_j for each neuron $j \in F_2$, where the fuzzy AND operation $a \wedge b = min(a, b)$, $|A| = \sum a_i$. The constant $\gamma^{ci} \in [0, 1]$ is acting as a choice parameter to emphasize or not the importance of the field ci, and α^{ci} used to prevent division errors.

$$t_j \in T = \sum_{i=1}^{k} \gamma^i \frac{|x^{ci} \wedge w_j^{ci}|}{\alpha^{ci} + |w_j^{ci}|} \tag{2}$$

The Equation 2 is described by (CARPENTER; GROSSBERG; ROSEN, 1991; CARPENTER; GROSSBERG; REYNOLDS, 1991; TAN, 1995; CARPENTER; GROSSBERG, 1988; TAN, 1992; TAN, 2004; WANG; TAN, 2015) as a fuzzy ART I operation. It suffers from the over generalization problem, where the input x^{ci} can not be distinguished coherently from the neuron weight w_j^{ci}. Considering $|x^{ci} \wedge w_j^{ci}| \leq |w_j^{ci}|$, there can be a match or a higher value of similarity from completely different stimulus just because they have the same length. It can generalize large amounts of data, but the lack of precision does not make it profitable. To solve this problem, Equation 3 was used by the authors in (WANG; TAN, 2015), and it is called fuzzy ART II.

$$t_j \in T = \sum_{i=1}^{k} \gamma^i \frac{x^{ci}.w_j^{ci}}{||x^{ci}||||w_j^{ci}||} \tag{3}$$

The operation $a.b$ is the dot product between two vectors defined as $\sum a_i b_i$ and the $||a||$ is the euclidean norm defined as $\sqrt{\sum a_i^2}$. This second form for checking the similarity is more consistent and it suffers less from over generalization. If analyzed in details, it calculates the $\cos(\theta)$ from the stimulus in x^{ci} and the neuron weight w_j^{ci}. Despite being more accurate than the fuzzy ART I for matching, it considers the angle between x^{ci} and w_j^{ci} and thus it can output higher values of similarity from different vectors because they have the same direction.

2) Inhibition: The code inhibition, in Equation 4, is used to select the most promising neuron $t_J \in F_2$ as a possible match between the presented activity in relation to all other neurons.

$$t_J = \max\{t_j : \text{for all } t_j \in T\} \tag{4}$$

3) Readout: A readout or output value, computed by Equation 5, is a prediction of the network upon each input presentation in I.

$$x^{ci} = x^{ci} \wedge w_J^{ci}, ci = 1, ..., k \tag{5}$$

The values computed by Equation 5 represent a generalized category close to the received stimulus and the codified weight in the neuron J. It can be used as the answer, prediction or classification given an activity x^{ci}. Another kind of readout can also be

performed by following Equation 6, thus generalizing between the stimulus and the selected neuron J.

$$x^{ci} = w_J^{ci}, ci = 1, ..., k \tag{6}$$

4) Resonance checking: The resonance value $m_J^{ci} \in [0, 1]$ is calculated by Equation 7 and it does another verification inside the selected neuron t_J to guarantee the similarity. But it considers the existing difference when $|x^{ci}| > |w_J^{ci}|$ in relation to X instead of W as presented by Equation 2. To be in resonance, all m_J^{ci} for $i = 1, ..., k$ inside the neuron J need to be greater than a precision value called a vigilance measured by $\rho^{ci} \in [0\ 1]$, otherwise a reset occurs. The reset just set $t_J = 0$, preventing it to compete again, thus a new neuron can be checked inside T.

$$m_J^{ci} = \frac{|x^{ci} \wedge w_j^{ci}|}{|x^{ci}|} > \rho^{ci} \tag{7}$$

In order to represent a resonated neuron, a boolean $y_j \in [0, 1]$ is stored inside the $Y \in F_2$ vector, where each neuron j has one y_j associated with it. When $y_j = 1$, then the neuron j is resonating, otherwise $y_j = 0$. The process of checking the resonance could soften or alleviate the existing over generalization problem into the inhibition and readout steps, but it is used if the network is in learning mode, in other words, if it needs to write data directly inside the neurons.

5) Learning: If the neuron J passes through the resonance checking it can be used to learn the new stimulus received from X. All the $w_J^{ci} \in W_J$ needed to be adjusted by Equation 8 is known as a fuzzy ART I learning method, where $\beta^{ci} \in [0, 1]$ is the learning rate of the field ci. This method produces a neuron that can be, again, over generalized, because it is updating the old weight w_J^{ci} by values that will always be less or equal $|w_J^{ci}|$, thus never fully representing the stimulus inside x^{ci}. This implies that the neuron can always be generalized but never specialized as pointed out by (TAN et al., 2008; TAN, 1992; TAN, 2004; WANG; TAN, 2015).

$$w_J^{ci} = (1 - \beta^{ci})w_J^{ci} + \beta^{ci}(x^{ci} \wedge w_J^{ci}) \text{ for all i = 1, ..., k} \tag{8}$$

To alleviate the over generalization problem in the learning step, the fuzzy ART II operation in Equation 9 presented by (WANG; TAN, 2015) can be used instead. This

fuzzy ART II operation guarantees the raw integrity of the received stimulus when codified into the neurons weights w_J^{ci} for all $i = 1, ..., k$.

$$w_J^{ci} = (1 - \beta^{ci})w_J^{ci} + \beta^{ci}x^{ci} \text{ for all i = 1, ..., k} \tag{9}$$

4.2 Perfect Miss Match

The *Perfect Miss Match* is a situation where the predicted stimulus comes from a neuron that is not suitable to represent it. It is important to use this method of detecting miss matches when the implemented model that uses a ANN needs to guarantee that each instance of a field i can be part of exactly one and only one neuron. Its detection occurs when a prediction, where the m_J^{ci} for the selected field i, is equals to 1, and for all the other fields, m_J^{ck} for all k are below of the specified ρ^{ci} parameter. After the detection of a *Perfect Miss Match*, a ANN can handle it with a special learning method called *Overwrite* described by Equation 10.

$$w_J^{ci} = x^{ci} \text{ for all i = 1, ..., k} \tag{10}$$

This learning method replaces the method described by Equations 8 and 9, it also guarantees that each distinct instance of a field i will belong to 1 neuron, thus turning a ANN suitable to store function values.

4.3 Adaptive Vigilance

A vigilance parameter is what enables the ANN to perform the *Resonance checking* step from its routine and identify the similarity of the received stimulus and the neurons weights in each field separately. This routine is controlled by ρ^{ci} parameters following Equation 7. The problem of using this method turns up when performing the *Resonance checking* step, neurons that are unsuitable to represent X are identified more often as viable to be used by the *Learning* step. To tackle this problem, the *Adaptive Vigilance* can be used to tell to the searching process that better solutions could be found by turning the selection process tighter and forcing the network to either select a non commited neuron or to detect a *Perfect Miss Match*. It's dynamics are controlled by raising the vigilance

parameter with Equation 11 if $m_J^{ci} > \rho^{ci}$ and if the neuron J does not resonate,

$$\rho^{ci} = \rho^{ci} + alpha_{vigilance} \tag{11}$$

where the variable $alpha_{vigilance}$ is a raising step slightly higher than 0. This process can also be done by a scaling term with Equation 12 if preferred.

$$\rho^{ci} = \rho^{ci} + \rho^{ci} * alpha_{vigilance} \tag{12}$$

If a resonating neuron could not be found by the end of the *Resonance checking* process, this method will force the Fuzzy ANN to detect a *Perfect Miss Match* or to create a new neuron for learning since at the end of its execution the selected parameter ρ^{ci} will converge to 1.

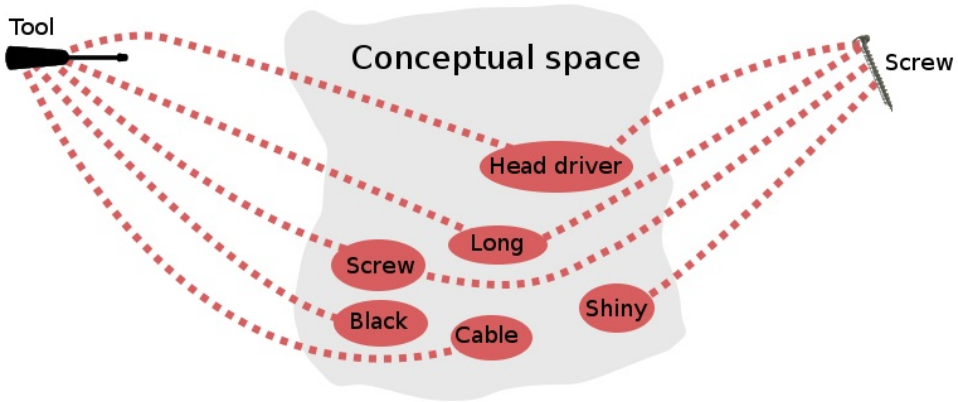
5 THE HONING THEORY

In this chapter, it is presented The Honing Theory os creativity and how one of its aspects, the surprise, can be represented by a Bayesian surprise model. The Honing Theory is used in this research as a basis which the HARP was built and it helps the deployed agents to generate creative strategies during the POMDP optimization process with the concept of surprise and novelty.

The Honing Theory describes creativity as the interaction between a human and its environment, where its capability in generating ideas will adapt according to experience. Furthermore, it also describes that a creative idea is conceived by a search procedure composed by two phases: *Associative*; and *Analytic*. This searching procedure is called associative thinking and analytic thinking, respectively, and they permit to activate concepts or create new ones representing creative ideas (GABORA, 2010a).

The process described by The Honing Theory assumes that concepts are composed by microfeature, where each microfeatures, inside the conceptual space is represented by interconnected neurons clusters, in the brain, that are called *Neural Cliques*. For instance, as showed in Figure 6, the concept of a tool is represented by the *Neural Clique* composed of the microfeatures: *Black*, that represents its color, *Screw*, representing its external relation, *Long*, that represents its size, *Cable*, representing it has a cable, *Head Driver*, that represents is ability to handle screw heads. This conceptual space can be compared to the ART's mechanism used to represent symbols, thus relating models and representing reality.

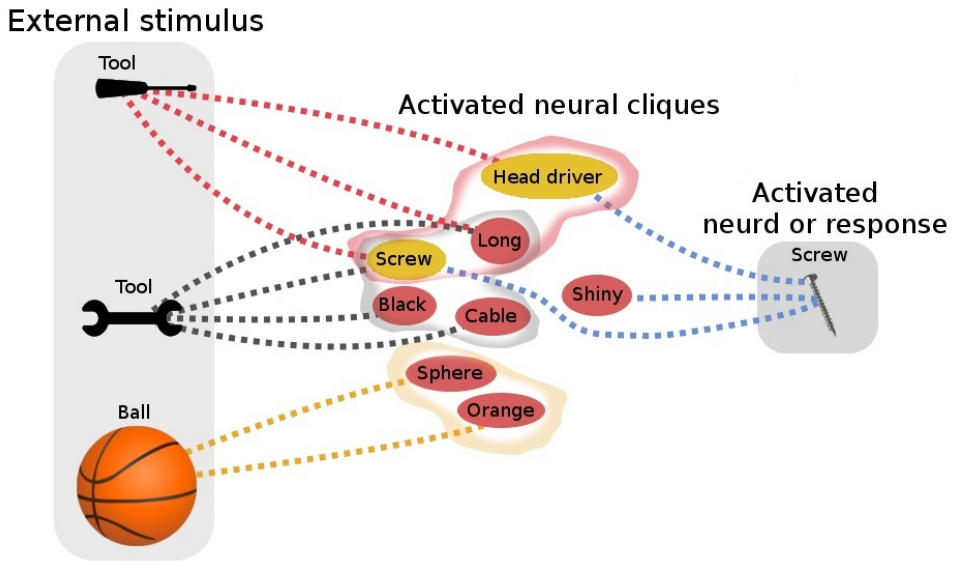
Figure 6 – The Honing Theory conceptual space.



Source: By the Author.

The associative thinking, according to The Honing Theory, permits the brain to activate neural cliques that can be interrelated through microfeatures, as the tool and screw neural cliques depicted in Figure 6, and it occurs until the brain converges into a single neural clique as a response. During the associative thinking, the brain can activate its analytical thinking to analyze the information inside the already activated neural cliques, thus helping to decide what cliques it should activate next. This ability to alternate between phases is defined as a *Contextual Focus*, where the brain decides between staying focused on various cliques simultaneously, in order to activate more of them, or in focusing into a single clique, helping to determine what it should do with the already activated ones. Each clique obtained as a response is called *neurd*.

Figure 7 – The Honing Theory process as a neural clique expansion.



Source: By the Author.

For instance, the idea about a *screw* at the right side of Figure 7 was generated by firstly receiving an external stimulus. This external stimulus was obtained perceiving objects, from an environment, them decomposing their essence into microfeatures. All microfeatures related to those objects were activated inside the brain and after their activation the neural cliques can be identified as the red, gray and brown transparent hulls. The set of all activated cliques is called *World View* and it represents what the brain is understanding at the moment of the observation. Those neural cliques are memories retrieved by an associative phase, consequently an analytical phase is activated to identify which cliques should be also activated or inhibited. For this example, the analytic phase

identified that the microfeatures *Head driver* and *Screw*, illustrated in yellow, are shared between other neural cliques, that represents other concepts, thus it decided to propagate their activation into them to see if something new can be discovered. This new clique, representing a *Screw*, obtained from the process is called *neurd* and it is used to compose a creative idea. This idea is assembled fusing the set of the received external stimulus, that shares microfeatures with the activated *neurds*.

5.1 Potentiality State

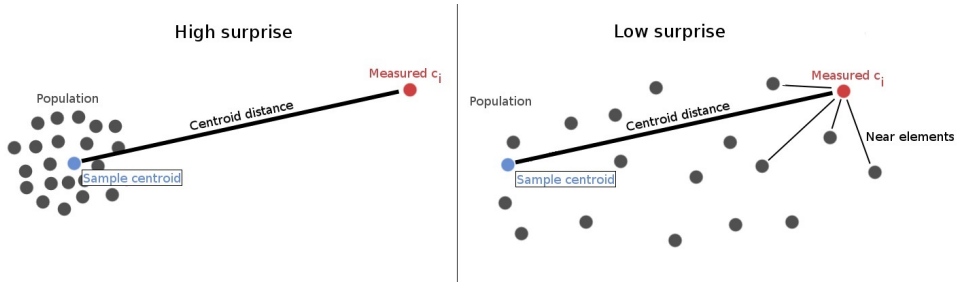
During the *Contextual Focus*, various neural cliques can be activated or inhibited and all the options that are available constitute a set of possible cliques to handle. This state, where many options are available, is called by the theory as a *Potentiality state* and it is governed by the brain's analytic phase relying mechanisms (GABORA, 2010a). Unfortunately, there are no explanations from the theory on how exactly it works inside the human brain, thus turning harder to create a computational model that can be directly mapped on top of a neural network or an ART system. Nevertheless, it seems that a decision making algorithm and an optimization method, as the one presented in Section 6 from the FALCON architecture, should help in solving the problem by experimenting an idea, seeing what will happen and evaluating it, thus associating the obtained value to its relying clique in order to facilitate decision making. Unfortunately, experimentation can handle utility values, thus the process can know if an idea is good or bad, but it does not measure how much that idea is unusual or novel in order to be classified as creative or not.

5.2 Bayesian surprise as a novelty metric

Creative ideas have to be novel and appropriate (BODEN, 2004). In order to measure novelty, there are few metrics based on concepts of surprise, unexpectedness, expectation that are commonly used (GRACE; MAHER, 2014). There are no consensuses on how to measure the novelty factor of a concept in relation to others inside a conceptual space, but the notion of surprise, novelty, can be used as a spacial distance that will tell how much distant the observation is from what is already known. As shown by the left box in Figure 8, the distance from a population's centroid can certainly be used as a novelty metric, because it is measuring how far the observed dot c_i is from others. However, this solution does not represent a true novelty since the dispersion of elements on that

population can influence how far their elements will be from the measured observation c_i, as shown by the right box in Figure 8. To tackle that problem, the Bayesian surprise calculates the spacial distance from objects considering variance of a population, measuring the novelty of an observation.

Figure 8 – Bayesian Surprise concept as the distance from a sample's centroid considering dispersion.



Source: By the Author.

5.3 Bayesian surprise

In order to a metric to be applied over an idea or a concept from a conceptual space, first they need to be transformed into a feature vector. Considering all the similarities of a conceptual space with a symbolic representation and definition from Chapter 4, all concepts from a conceptual space are treated as symbols, thus they can be directly assembled in a feature vector to compute any spacial metric. With the numerical representation of concepts, the Bayesian surprise (BALDI; ITTI, 2010) can evaluate how much a feature from its feature vector $C = \{c_i, ..., c_n\}$, being n the total number of features, is different from an already established population $P = \{PE_1, ..., PE_m\}$, where each $PE_i \in P$ is a vector of the form $PE_i = \{p_1, ..., p_n\}$. In its simplified form, it is defined by Equation 13, where c_i^{new} is the feature i from C that will be compared, σ_i^2 the variance of the feature i in P and $\bar{m}_i$ being the arithmetic mean of the feature i from P.

$$S(c_i^{new}) = \frac{N}{2\sigma_i^2}[\sigma_i^2 + (c_i^{new} - \bar{m}_i)^2] \tag{13}$$

When computing the surprise factor or how novel a feature is compared to what is already known, it seems feasible in using the Bayesian surprise since it considers the distance as $[\sigma_i^2 + (c_i^{new} - \bar{m}_i)^2]$, however a scaling factor described as $\frac{N}{2\sigma_i^2}$ is also used to scale its results based on the inverse of the variance from a population. By accomplishing

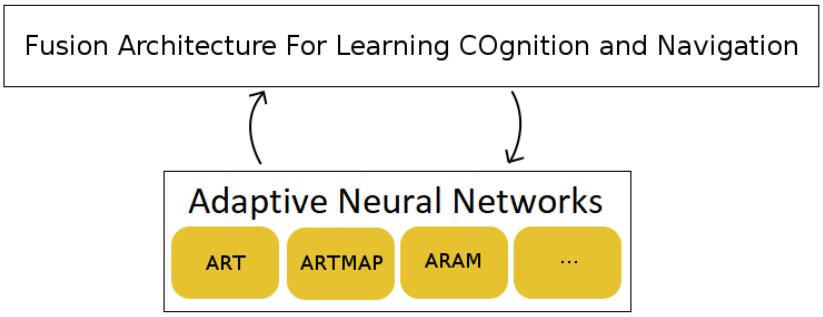
that, the computed surprise will be scaled to receive even higher values when considering a high variance and in low values when considering a low variance, where the variable N is used to define the starting point or number in which the measured distance will start to be scaled up or scaled down. This method is used in this research to compute not only the utility value from experimentation considering an environment e, from a simulated POMDP, but to calculate the novelty of a decision.

6 FUSION ARCHITECTURE FOR LEARNING COGNITION AND NAVIGATION

This last background chapter presents the necessary background to understand the FALCON that is used to permit an agent to reason about what it is experiencing and to recognize the existence of the *Time Flow*. This architecture is used in this research to simulate a POMDP that will help the agent to perform its creative reasoning.

The Adaptive Neural Networks, as illustrated in Figure 9, is any neural network that works with representative fields, as discussed in Section 4. There are some main ANNs referenced in this work. They are the ART system, ARTMAP and ARAM. The deployed solutions for all HearthBots uses aspects of each one of them. Those aspects, that involve mapping the input stimulus into fields, learning and adapting them, are discussed in Section 4.

Figure 9 – Adaptive Neural Networks and FALCON interaction.



Source: By the Author.

The FALCON is used to enhance the representation, of stored information inside an ANN, to simulate a POMDP. As showed in Figure 9, a FALCON architecture communicates with an ANN to realize how information is being stored inside of it. It can be interpreted as a reinforcement learning handler, while a ANN handles the supervised and unsupervised learning. For instance, the FALCON architecture can be described as reactive, that senses and responds based on information gathered from two States, the present and immediate future, that considers as many future States as it wants in order to take a decision through its mechanisms. The decision taken will tell to the system how it should handle information to reason and how to maintain the integrity of the simulated POMDP. It is important to note that, the FALCON architecture can be deployed with other kinds of neural networks and is not limited exclusively to the ANN family.

A FALCON is an architecture that enables an agent to sense, think and act in a dynamic environment. It is assembled with three fields on the F1 layer of the ANN, E, A and R. Where E represents all the environment, A all the possible actions that the agent can perform and R the reward received by the agent when performing the action expressed by A. The environment E is composed by $\{e_1, ..., e_n\}$, where n represents the total amount of variables used to describe the environment. According to the FALCON architecture, an action is described by a binary array composed by m actions, thus A needs to be in a binary array form, where $A = \{a_1, ...a_m\}$, $a \in [0, 1]$, and m is the total amount of actions that the agent can perform. The R field is used to store a reward, and is typically constructed with the form $R = \{r, 1 - r\}$, where r is the immediate reward received after performing an action represented by A, and $1 - r$ the complement of r. The complement of the reward is used in a FALCON architecture by the reactive and temporal interaction models proposed by (WANG; TAN, 2015) and described in details in Sections 6.1 and 6.2.

6.1 Reactive Model

The reactive model consists in sensing the environment through each input signal $e_i \in E$, and then select an action $a_i \in A$ to be performed in search of a reward r. It is divided into two parts, where the agent firstly seeks for a valid action to be performed, *From Sensory To Action*, and secondly when the agent performs the selected action and then presents the environment feedback to the ANN to be learned, *From Feedback To Learning*.

6.1.1 From Sensory To Action

In order to select an action, an agent presents its current view of the environment in the form of a vector to the ANN. This environment vector is coded inside a field from the $F1$ feature layer. After building the environment vector inside E, the agent needs to speculate what he requires to accomplish given an observation of an environment E. In order to tackle that, it asks for the ANN what action to perform through a *Direct Acess Method*.

The *Direct Acess Method* configures the action vector $A = \{a_1, ..., a_m\}$ with each $a_i \in A = 1$. This action vector A, from the *Reactive Model*, represents an action mask that

codes all the possible actions that have been executed since the first observation of the environment E. The same method is used to build the reward vector R, where $R = \{1, 1\}$.

After using the *Direct Acess Method*, the agent presents the three coded vectors, E, A and R to the ANN as fields from the $F1$ layer to fulfill a prediction. The coded vectors will force the network to predict an action and reward vector that will subscribe the previously configured vectors A and R. The action a_s that will be performed by the agent is finally obtained from the action competition method following Equation 14.

$$a_s = \max\{a_i : \text{for all } a_i \in A\} \tag{14}$$

6.1.2 From Feedback to Learning

After executing the *From Sensory To Action* part, the agent then performs the selected action a_s and waits for the environment to respond. As specified by the FALCON algorithm, the agent needs to wait the environment in which it is interacting to respond. This could be wasteful, considering that it could take this extra time to perform other tasks. A response from the environment is coded inside a variable r_e and then it needs to be interpreted, and coded as a reward to be stored in r, by an objective function f that needs to be specified by the programmer. This interaction will generate a new environment E_{new}, assuming that the action performed by the agent can eventually modify the present environment E. Finally, the agent needs to code the old environment E, the performed action a_s and the received reward r into E, A and R for learning, where E takes no change, $A = \{a_1, ...a_m\}$, where each $a_i = 0$ and $a_s = 1$, and $R = \{r, 1 - r\}$. Those three coded vectors are presented to the ANN for learning.

6.1.3 Reinforcement learning

When learning in the *Reactive Model*, the ANN can use two kinds of reinforcement learning techniques. One of them is represented by the Fuzzy ART operations that stamps the stimulus from E, A and R inside the selected neuron from the learning step described in Section 4 and the second one is used to control the quality of a decision from the agent based on the predicted action a_s that will be performed on the environment E. This second form of reinforcement learning is divided into: 1) *Neuron Erosion*; 2) *Neuron Reinforcement*; and 3) *Neuron Decay*. Each part of the reinforcement procedure relies on a

confidence vector C, where $C = \{c_1, ..., c_n\}$, being n the total number of neurons. Each of the $c_i \in C$ is a value that ranges inside $[0, 1]$, being 0 a neuron with 0% confidence and 1 being a neuron that has a 100% confidence that its response is accurate.

6.1.4 Neuron Erosion

The *Neuron Erosion* is applied when the reward r, interpreted by the objective function f, is lower than a previous r_{old} obtained and stored from the immediate past action performed by the agent. The erosion tells that the performed action a_s, selected from the *From Sensory To Action*, caused a negative impact on the overall agent's rewards over time. This step, as being reactive, calculates the erosion considering one interaction in the past. After the detection of an erosion with the condition $r \leq r_{old}$, the learned action mask A, from the *Feedback to Learning* step, suffers a reset and is configured as $A = \{a_1, ..., a_m\}$, where each $a_i = 1$ and $a_s = 0$. This reset configuration associates a mask with E that forces the agent to explore other actions when facing the same environment overtime, due to the fact that the action a_s is selected with the *max* operator by the Equation 14,

$$c_J = c_J - c_J * \alpha_{erosion} \tag{15}$$

where the variable $\alpha_{erosion}$ represents the decay rate of the confidence level for the represented neuron J.

The reward field R suffers the same reset configuration, where $R = \{1 - r, r\}$. It is essential to note that the reset configurations inside A and R replace the A and R inside the same learning method from Section 6.1.2. After that is done the recently computed vectors A and R alongside E are presented to the ANN for learning. After the learning gets completed, the selected neuron J from the *Inhibition* step from the categorization mechanism in Section 4.1 is used to update its correspondent inside the confidence vector C with Equation 15.

6.1.5 Neuron Reinforcement

Neuron reinforcement occurs when a temporal change happens in r, after receiving a positive environment feedback. To calculate that temporal change, the variable r_{old} is used on the condition $r > r_{old}$, as in the *Neuron Erosion*, and it is also calculated based on

the immediate past action performed by the agent. Differently from the *Neuron Erosion*, the reinforcement updates the confidence vector C and leaves the learning process as presented in Section 6.1.2. A reinforcement is done after presenting the vectors E, A and R to the ANN for learning. The selected neuron J, also from the *Inhibition* step from the categorization mechanism in Section 4, is used to update its correspondent inside the confidence vector C with Equation 16.

$$c_J = c_J + c_J * \alpha_{reinforce} \tag{16}$$

As in the *Neuron Erosion*, the variable $\alpha_{reinforce}$ represents the rate of change in confidence on the selected neuron J.

6.1.6 Adaptive Cognitive Code Pruning

An adaptive neuron pruning occurs when the confidence level of a neuron c_i is below a threshold $t_{pruning}$. This threshold is given a priori and represents the minimum level of confidence that a neuron needs to have in order to be viable. If the neuron i confidence $c_i < t_{pruning}$, then the neuron is marked to be removed or reused by the network. Considering that neurons confidence suffers changes by the learning procedure, the neuron pruning could occur a few times depending on how well the agent is doing in an environment. This implies in the usage of the *Neuron Decay*, a function that decreases periodically the confidence level of every neuron represented inside C. The *Neuron Decay* is calculated by the Equation 17, and it permits the pruning to occurs often,

$$c_i = c_i - c_i * \alpha_{decay} \texttt{ forall } c_i \in C \tag{17}$$

where the variable α_{decay} represents the decay rate of each neuron. It is important to note that the neuron pruning relieves the network from non-reliable neurons, that could lead the agent to a bad decision when selecting an action a_s, and also reduces the size of the network in terms of space occupied by all the stored neurons.

6.2 Temporal Difference Model

A temporal difference model inside a ANN was firstly introduced by (WANG; TAN, 2015), where the authors propose a FALCON architecture that incorporates the

Q-Learning mechanism inside its dynamics. This model was incorporated to deal with the problem of delayed rewards, received several steps in the future, that the *Reactive Model* could not deal with since it calculates the temporal difference based on one step to adjust the neurons confidence level. When performing with Q-Learning, the FALCON model codifies each environment as the *Reactive Model*, but it uses one neuron per action instead of an action mask. With this representation, the Q-Learning method can be applied directly to the FALCON model to calculate Q values with the function $Q(s, a)$, where s represents the current environment and a the action that was performed on s. Each $Q(s, a)$ replaces the immediate reward inside the R field and its value represents an entrance on a virtual Q-Learning table created by the network's neurons.

6.2.1 From sensory to Action with Q-Learning

The prediction process for the *Q-Learning* method starts by presenting to the ANN the environment vector E that represents the state s, the selected action vector $A = \{a_1, ..., a_m\}$, where m is the total number of available actions, $a_i \in A, a_i = 1$ and for each other $a_j \in A | a_j \neq a_i, a_j = 0$, and $R = \{1, 1\}$ with the *Direct Code Access* method to request a prediction. This way of coding fields inside the $F1$ layer of the ANN codes each neuron as a virtual state of a *Partially Observable Markov Decision Process*, thus being possible to store each $Q(s, a)$ as transition values from the current state s to a new state s'.

The $Q(s, a)$ function value is obtained from the predicted reward field during the presentation of E, A and R. This predicted $Q(s, a)$ value is then used to select an action $a_s \in A$ to be performed by the agent through an *Action Selection Policy*, thus resulting in a direct interaction with the environment E during its presentation. That interaction leads to a reward, response from the environment, that is used to update the recently predicted $Q(s, a)$. It is important to note that, predicted $Q(s, a)$ values obtained from non-committed neurons, blank neurons inside the ANN, need to be initialized for the algorithm to work properly. The initialization is accomplished by Equation 18, where each $Q(s, a)$ from non-blank neurons is assigned with the value $\frac{1}{2}$,

$$Q(s, a) = \frac{r_1}{\sum_{i=1}^{k} r_i} \tag{18}$$

where $r_1 \in R$ is a recently predicted $Q(s, a)$ and $r_2 \in R$ the complement, described as

$1 - Q(s, a)$, of $Q(s, a)$ to be used as a flag that enables the identification of a prediction from a non-committed neuron, and k being the total amount of variables inside the field. For instance, if a predicted $R = \{1, 1\}$, then the calculated $Q(s, a)$ will be initialized with the value $\frac{1}{2}$ as already pointed out.

6.2.2 Value Estimation

A value estimation is what permits the ANN to calculate temporal differences from a $Q(s, a)$ to $Q(s', a')$, where s is the current environment, a the action that was performed on s, s' the generated environment after performing the action a and a' the action selected to be used as a guess during the presentation of s'. This guess represents what future action the agent want to use in order to calculate its temporal difference, thus the result of the algorithm will rely upon the *Action Selection Policy* for a and a'. This method enables the agent to update its virtual Q values table, thus it can be considered as a complement learning operation from the learning process in a Temporal Difference FALCON. All estimations from a temporal difference are calculated with Equation 19,

$$\Delta Q = \alpha * TDerr \tag{19}$$

where $\alpha \in [0, 1]$ represents the Q-Learning rate and $TDerr$ being the temporal difference error calculated with Equation 20. The $TDerr$ can be calculated by two ways, the *Q-Learning* and the *State Action Reward State Action* SARSA, both are presented below.

Q-Learning: The temporal FALCON method relies on estimating the $Q(s, a)$ with the assistance of an *Action Selection Policy* to select the action a to be performed during the presentation of the environment s. After performing the selected action a, the agent will cause a change on the environment s that will lead to a s'. This s' is then, used to estimate the future $Q(s', a')$. The a' is selected in a such a way that maximizes $Q(s', a')$ to the highest value possible, this can be seen as the max operator from Equation 20,

$$TDerr = r + \gamma * \max_{a' \in A'} Q(s', a') - Q(s, a) \tag{20}$$

where r is the immediate reward received from the environment after the execution of the selected action obtained from the process described in Section 6.2.1, $\gamma \in [0, 1]$ being a discount parameter that scales the temporal difference, and $\max_{a' \in A'}$ the best action a' to be performed during the presentation of the environment s'.

SARSA: The SARSA method, represented by Equation 21, is based on changes caused by actions selected from an *Action Selection Policy*. This implies on not selecting a' based on a max operator as showed in Equation 20. Instead, it will always calculate $Q(s', a')$ based on a selected action that will be performed during the presentation of an environment s',

$$TDerr = r + \gamma * Q(s', a') - Q(s, a) \tag{21}$$

where r is also the immediate reward received after a being performed, as in the *Q-Learning*, and $\gamma \in [0, 1]$ the discount parameter to scale the temporal difference. The SARSA method is also called the On-Policy stemmed from the fact that it will calculate the temporal difference based on the decision of an *Action Selection Policy*.

6.2.3 Bound rules

Q values stored inside neurons may be outside of the limits, typically from the working interval $[0, 1]$ in ART systems, and thus bounding rules need to be applied. Two common rules can be used: *Threshold* and *Self Scaling*. The *Threshold* rule uses the Equation 22 to limit the calculated Q values inside the working interval.

$$Q(s, a) = \begin{cases} 1 & \texttt{if } Q(s, a) > 1 \\ 0 & \texttt{if } Q(s, a) < 1 \\ Q(s, a) & \texttt{otherwise} \end{cases} \tag{22}$$

The problem with the *Threshold* is that occasionally a meaningful Q value can went out of its bounds, thus generating overgeneralized responses based on ambiguous Q values. To solve this, the *Self Scaling* could be used instead, where the Q values will be self scaled by Equation 23.

$$Q(s, a) = \alpha * TDerr * (1 - Q(s, a)) \tag{23}$$

This method can be applied in both Q-Learning value estimation methods, Q-Learning and SARSA and it will force $Q(s, a)$ to a self-scaling value inside the working interval without cutting meaningful Q values out of the bound. It is important to note, as said by

(WANG; TAN, 2015), that the predicted Q values need to be from the network and not from an external data structure to ensure its proper learning.

6.2.4 From Feedback to Q-Learning

The learning process is accomplished by presenting to the ANN the updated $Q(s, a)$ obtained from Equation 24 along side with the environment vector E that represents s and the action vector A assembled to represent the selected action a,

$$Q(s, a) = Q(s, a) + \Delta Q \tag{24}$$

where s is the current state in case of *Q-Learning* and the old state in case of SARSA, the selected action vector $A = \{a_1, ..., a_m\}$, where m is the total number of available actions, $a_i \in A, a_i = 1$ and for each other $a_j \in A | a_j \neq a_i, a_j = 0$, and $R = \{Q(s, a), 1 - Q(s, a)\}$. For instance, the $Q(s, a)$ complement inside the R field is used to identify a prediction from a non-committed neuron as already pointed in Section 6.2.1. The R field is configured to hold the recently calculated Q value to be learned using the ANN five step routine described in Section 4.

Part II

Creative agent reasoning through adaptive neural networks

7 COMPUTATIONAL MODEL OF THE HONING THEORY AND THE HONING ADAPTIVE RESONANCE PROCESS

In this Chapter, the proposed symbolic computational model of The Honing Theory is presented. To first define how a computational system of it should behave, it is important to know how to represent information from its conceptual space. It stems from the fact that The Honing Theory operates within a conceptual space made of microfeatures. To tackle this problem, in this research a conceptual space should be assembled as a symbolic set, made of symbols, where each symbol is a string that represents an abstraction of an object in reality. The semantic representation is achieved by a Honing Network, that uses ontologies, based on semantic networks and frames, to represent semantics.

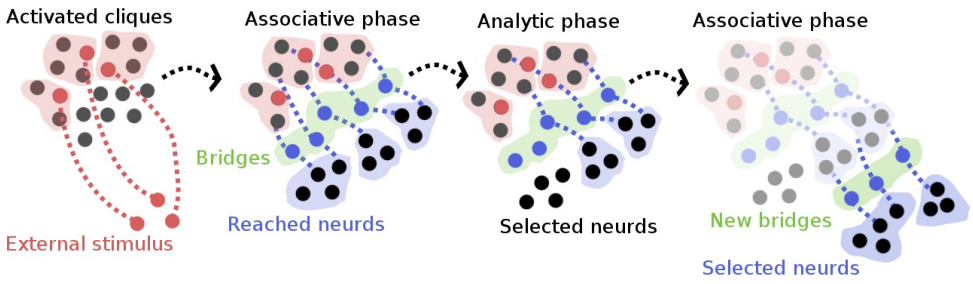
Both The Honing Theory and The Adaptive Resonance Theory describe how memories are retrieved and assembled from different points of view, however, they have much in common, for example, their neuron activation and retrieval mechanism. In this chapter, it is also proposed the HARP (Honing Adaptive Resonance Process), where creative ideas are generated with a FALCON architecture and two ANNs that are able to simulate aspects of proposed computational model of The Honing Theory to deploy emergent creative thinking into machines.

7.1 Conceptual space of vertexes for analytic and associative phases

The Honing Theory works inside a conceptual space made of symbols, that represents microfeatures, and its objective is to generate a creative idea. There are no current implementations of it to this date, and is proposed in this research that its process should be modeled and implemented as a searching heuristic that works with a semantic representation defined as a Honing Network, assembled with graphs, like semantic networks and ontologies. The proposed model consists in receiving data from an environment E, organizing that data into understandable symbols set $S = \{s_1, ..., s_n\}$ that represents the conceptual space, where $s_i \in S$ is associated with a vertex v_i from a Honing Network with a mapping function $f : V \rightarrow S$, and n being the total amount of microfeatures observed for all objects on E. As shown by the example depicted in Figure 10, it is proposed that the search procedure should work in associative and analytic modes that encompasses all the theory aspects on how an idea is generated.

The process showed in Figure 10 starts by receiving microfeatures as external

Figure 10 – Honing Theory proposed process with analytic and associative phases.



Source: By the Author.

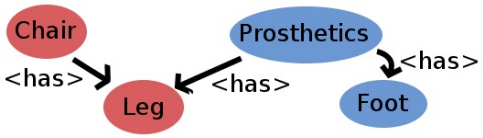
stimulus and activating vertexes from the conceptual space. The relying Honing Networks, that are directly connected to the activated vertexes, are fully retrieved as neural cliques and used to start the associative mode. As accomplished by the *Associative Phase* in Figure 10, the activated Honing Networks are used to trigger signals to vertexes from the conceptual space, inside a green hull, that represent a relationship between Honing Networks, those vertexes are defined as *bridges*. After activating all *Bridges*, new Honing Networks are retrieved as *neurds*. Moreover, when in the analytic mode, the activated *neurds* passes through a process of inhibition, and a few Honing Networks stay active. Those who stayed will be used to trigger a new associative phase as activated cliques. The process finishes when reaching a point without any bridge to activate or by a predetermined iteration limit.

7.1.1 Honing Network as unstructured information

An easy way to represent neural cliques are using semantic networks, where each semantic network can share its vertexes representing large amounts of information in a compact way. The problem when using a searching heuristic, that seeks by vertexes, to deal with such structures is the lack of a naive way in dealing with cycles and undirected edges. For example, on the graph depicted in Figure 11 representing a chair and a prosthetic leg, a possible searching, starting from the vertex associated with the symbol *chair* to reach *prosthetic*, could be accomplished with a *Depth First Search* starting at *chair* until reaching the bottom, at *Leg*, and then seeking for parents until reaching *prosthetics*. This kind of search procedure will contribute to the need of linking vertexes by undirected edges accessing parents, thus adding extra complexity in terms of space. Furthermore, if all edges get associated with a semantic stereotype, then an extra mechanism to establish

when to seek for parents will need to be implemented, since a computer can not distinguish what is a relevant semantic relation that will lead to a desired answer.

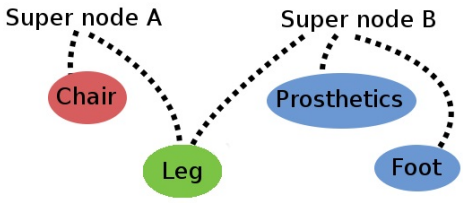
Figure 11 – Semantic network representing a chair and a prosthetic leg.



Source: By the Author.

Both can be tackled by searching procedures, as *Depth First Search* or *Breadth First Search*, with the help of markers defining when a cycle occurred and when to seek for parents. The problem of using such mechanisms, is the big effort in handling paths, created by the searching process, to reach vertexes from other semantic networks. To sidestep those problems, in this research it is proposed that the structure used to represent the conceptual space should be mix between semantic networks and an unstructured ontology. This structure is defined as a Honing Network and it is composed by super nodes without edges between children vertexes. For example, in Figure 12, the Honing Network is composed by two super nodes that represent contracted semantic networks representing concepts, where the *Super node A* indicates the chair semantic network, the *Super node B* represents the prosthetic leg semantic network, and all vertexes are floating inside a conceptual space. The super nodes from The Honing Theory are virtual and used by humans to understand its meaning when developing a system.

Figure 12 – Honing Network super node.

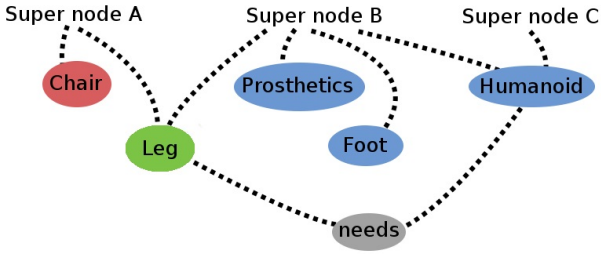


Source: By the Author.

The structure illustrated in Figure 12 permits a search procedure to seek vertexes in a linear way without data overhead, thus simplifying a system in terms of coding. Nevertheless, it does not indicate semantic relations between super nodes. In order to tackle that problem, semantic vertexes, called *bridges*, are used between super nodes

replacing stereotypes from semantic networks. For example, in Figure 13, the super nodes B and C has the *humanoid* vertex in common and it's linked to a vertex called *needs*, used to constitute a semantic relation between super nodes, where the *Humanoid needs* a *leg*. The essential problem here is how to identify which leg the humanoid needs, it could be the *chair* leg from the super node A or the *prosthetic* leg from super node B. In this research is defined that the related super node, or the one that will be activated, is the one with the most or some microfeatures in common. In this case, the super node B will be related with C, because they have the vertex *humanoid* in common.

Figure 13 – Honing Network interconnected super nodes.



Source: By the Author.

By using the *bridge* vertexes, the semantic relations between concepts in a conceptual space can be represented. The principal problem that arises with this representation is the lack of a way to distinguish vertexes between super nodes. For example, in Figure 13, the super node B constitutes a prosthetic leg that can be used by a humanoid, thus it does not require a leg as represented by the bridge *needs*. There are two ways to tackle this problem, the first one is to use one bridge set for each super node and the other one is on creating an exclusive set of vertexes for each super node. Either solution will enable to identify correctly, vertexes and the semantic relation between super nodes.

7.1.2 Activation

Any vertex $v_j \in V$ can be activated from external stimulus if obeying the activation function $\Phi(s_i)$, where s_i is the received symbol, representing its microfeature, from a vertex $v_i \in V | i \neq j$. The set $A = \{v_1, ...v_n\}$, represents all activated vertex from a received external stimulus that will activate the first set of neural cliques $N = \{C_1, ..., C_m\}$, where m is the total of neural cliques that were activated and each $C_i \in N$ represents a Honing Network. Each vertex can be activated by two types of connections: 1) Direct; 2) indirect.

Direct connections describe vertexes from Honing Networks activating vertexes from other Honing Networks. On the other hand, indirect connections are accomplished by *bridges* that interconnect Honing Networks with semantic relations. Connections made by *bridges* are used to activate Honing Networks that are bound with a semantic relation, moreover, *direct* connections are used to describe what microfeatures each activated Honing Network has in common. For instance, the example depicted in Figure 14 shows a direct connection between a chair and a prosthetic leg and also a indirect connection between an amputee and the prosthetic leg Honing Network. This indirect connection is made by a bridge, shown as the gray node, and the indirect connection is accomplished through the *humanoid* and *knee* nodes. It is important to note that the super node from the leg microfeature, illustrated in Figure 14, is not shared between super nodes.

Figure 14 – Direct relation between a chair and a prosthetic leg.

Source: By the Author.

7.1.3 The Honing Theory algorithm as a GRASP process

A Greedy Randomized Adaptive Search Procedure (GRASP) process is a meta-heuristic used for generating solutions to optimization problems (BINATO; OLIVEIRA; ARAUJO, 2001) (FEO; RESENDE, 1995), where in this research is used to develop a computational model for The Honing Theory searching process described in Section 7.1. This model facilitates the creation of an inference engine that can handle sets of Honing Networks, thus being able to generate creative ideas as a set of activated cliques and expanded neurds. The GRASP is described as a two phase process as follows: 1) Develop

initial solutions, where initial solutions are assembled and grouped to be latter explored; 2) search a local enhanced solution, where the assembled solutions from phase 1 are used to achieve an unique solution.

The Algorithm 1 is responsible for the first phase of the proposed Honing Algorithm based on GRASP, where microfeatures are activated from an external stimulus vertex s from a Honing Network. By receiving the external stimulus, bridges are activated and stored inside B, where each $b_i \in B$ is obtained from the *bridges* function. After obtaining all bridges, the algorithm employs them to activate parent vertexes that represent activated microfeatures from the Honing Network and stores each of them inside a set P. Each parent vertex inside P is used to retrieve a neural clique $IC_I \in C$, through *cliques* function, where each neural clique represents what neurds can be activated. A neurd is activated and stored inside a response clique set A, if $IC_i \cap S \neq \emptyset$. This response is finally returned by the algorithm to perform the honing procedure by Algorithm 2.

Algorithm 1: Greedy Randomized Construction pseudo code for associative phase clique activation

Input: Seed s
Output: Activated Cliques A

```
1: S ← activateClique(s)
2: bridgeResponse R ← ∅
3: bridges B ← bridges(s)
4: for each bridge b_i ∈ B do
5:    P ← parents(b_i)
6:    C ← cliques(P)
7:    for each clique IC_i ∈ C do
8:       if IC_i ∩ S ≠ ∅ then
9:          A ← A ∪ IC_i
10:      end if
11:   end for
12: end for
13: Return A
```

Source: By the Author.

The second phase of the proposed process, described by Algorithm 2, selects which cliques obtained from Algorithm 1, stored inside A, should be used as neurd responses, evaluating them with an evaluation function $CheckSolution$. If the examined cliques inside A are valid, then the *top-k* max responses from $CheckSolution$ will be selected, composing the final solution O, otherwise, invalid ones, will suffer a repair to be removed from the

process. The evaluation occurs with a creative assessment metric, as argued in Section 4, where a domain bounded utility value evaluation is given by a specialist and a novelty metric calculated as a Bayesian surprise between the examined solution and an already established population. The algorithm continues building its solution through extracting all vertexes from a valid A and using them at the next iteration of t as new bridges.

7.1.4 Discussion

The proposed model and algorithm, for The Honing Theory, enables to generate adaptive randomized constructions according to a GRASP process that represents the associative and analytic phases. This model can generate ideas considering not their evaluation, but also the relying process, what differs from most solutions in generating creative ideas. However, it has two major problems: 1) It relies in a preconceived Honing Network given by a specialist, thus it will be a symbolic system bounded to its domain; 2) the *CheckSolution* function, that evaluates the quality of generated ideas, commonly deployed as a value plus novelty metric measuring the creativity level of the partially generated solution inside O, will also behave within a specified domain since the system is symbolic. Considering those problems, an emergent model can be used instead, where creative ideas can be generated without a Honing Network, thus contributing to an emergent behavior during the interaction between an agent and the environment.

7.2 The Honing Adaptive Resonance process

The HARP is deployed with two ANN. One of them is used to store states to simulated a POMDP, mainly representing The Adaptive Resonance Theory, and it permits to perform an optimization technique to obtain maximum paths describing how an agent will perform a task through sequence of actions. The second one is used to compute how novel an action is for an observed state, mainly representing the creative thinking process described by The Honing Theory, thus it will be able to reach novel paths resembling the creative thinking concept. As illustrated in Figure 15, the HARP is represented with the POMDP and Novelty ANN. The shown example represents what an agent in seeing through an input signal Env, representing a POMDP state, and the action set $\{a_1, a_2, a_3\}$, representing all possible actions that an agent can perform. To decide what action it will take, all the action set and observed state are presented to both networks at the same

Algorithm 2: Honing algorithm based on a GRASP procedure

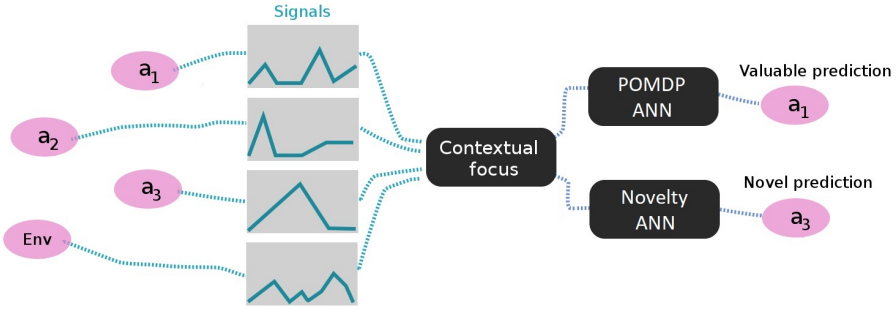
Input: Iterations t, Seed s
Output: Output cliques O

New seeds $S \leftarrow \emptyset$
for $t \geq 0$ **do**
  $A \leftarrow \emptyset$
  if $S \neq \emptyset$ **then**
    for each $s_i \in S$ **do**
      $A \leftarrow A \cup GRC(s_i)$
    end for
  else
    $A \leftarrow$ GRC(s)
  end if
  if (!CheckSolution(A)) **then**
    $A \leftarrow$ Repair(A)
  end if
  $S \leftarrow$ microfeatures(A)
  $O \leftarrow O \cup A$
  $t \leftarrow t - 1$
end for
Return O

Source: By the Author.

time. As a prediction, both networks will return an action that the agent need to perform considering the observations.

Figure 15 – The Honing Adaptive Resonance Process.



Source: By the Author.

A module called *Contextual Focus* is responsible to decide which response to use. If using the POMDP ANN response, the agent will perform an action that will maximize a POMDP path. On the other hand, the selected novel action, from the Novelty ANN, will lead to a novel path. The proposed solution establishes that creative thinking is achieved

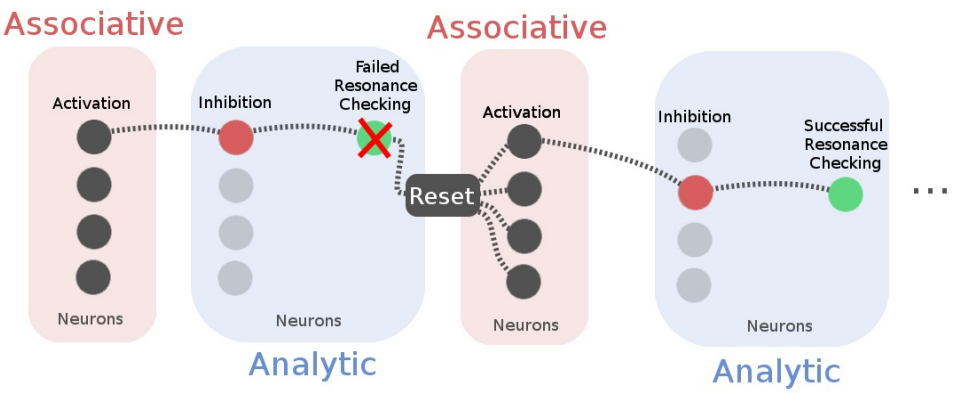
by the interaction of agent and its environment, where the optimization of all POMDP paths is influenced by novel decisions triggered by the *Contextual Focus* module.

7.2.1 The Honing Theory as an ART system

Most of The Honing Theory can be direct mapped on top of an ART system that simulates a POMDP, since both describe theoretical concepts that overlaps and represent how the human brain retrieves memories and form strategies. The main problem when representing The Honing Theory with an ART system is the lack of a way in discerning what are ideas, what step of an ART system is mapped as associative or analytic, what is considered a potentiality state, contextual focus, neural cliques and neurd recruiting.

Considering that The Honing Theory can be mapped into unique levels of meaning, it is proposed that The Honing Theory/ART mapping should be done in two domains, abstract and physical. The physical domain is directly tied to the working steps of an ART system, where contextual focus happens at each step transition from the five step routine described in Section 4. The neurons activation step triggers a readout that enables to retrieve memories, thus representing an associative mode, differently, inhibition and resonance checking steps evaluate neurons analytically. A resonance checking step can also trigger a new activation procedure, through a reset module, and consequently performing the same process described by the proposed Honing algorithm based on a GRASP procedure as showed in Figure 16, where associative phases, described by red hulls activate all four neurons available, as gray circles, an analytic phase selects one, as a recruited neurd, for resonance checking and a reset module triggers a new associative phase if resonance fails.

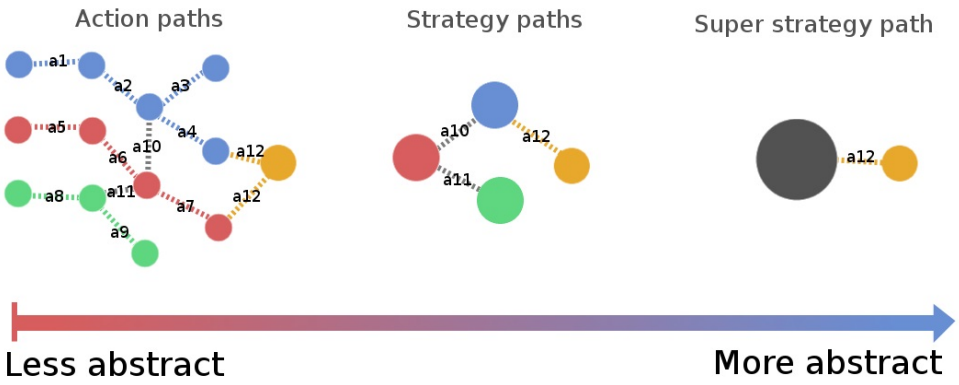
Figure 16 – The HARP physical domain as a cyclic process.



Source: By the Author.

In contrast to the physical domain, an abstract one deals with simulated POMDP paths that can be grouped by meaning level, where less abstract paths are closer to the physical domain. For example, Figure 17 shows three abstraction levels from less abstract on the left to more abstract to the right. Less abstract paths are created directly by performing single actions, from $a1$ to $a12$, where the blue, red and green circles sets represent different generated paths, being the orange circle a final state. On the other hand, paths can be grouped, as shown by the middle example depicted in Figure 17, where actions $a1$ to $a4$, $a10$ and $a12$ are grouped inside the blue circle, $a5$ to $a7$, $a10$, $a11$ and $a12$ grouped inside the red circle, and actions $a8$, $a9$ and $a11$ grouped inside the green circle. This representation assumes that the agent, when entering one of those states, will perform all of the grouped actions, and the virtual POMDP paths will represent links between previously established strategies. At least, a super strategy node is represented by linking all possible strategies, and one of them will be performed to reach the final state through action $a12$.

Figure 17 – Abstraction levels.



Source: By the Author.

Each domain, less or more abstract, are mapped as a conceptual space, where contextual focus and potentiality state happens at each decision step when selecting actions to perform. By supposing that, analytic phases will occur when selecting an action to perform and associative phases will occur when selecting a set of possible actions to explore.

7.2.2 Contextual focus for action prediction

In order to achieve associative and analytic phases inside an ART system, is proposed that a conceptual space be formed by a set of all states from a simulated POMDP inside a ANN, being each state represented by neuron weights from an environment, action and reward fields. With this conceptual space as ART neurons, a creative idea is obtained in the form of an *action* predicted by a FALCON architecture with more than two fields. This prediction represents something that can interact with an environment e and generate a response in the form of a reward, thus an optimization method, as the reactive or temporal Q-Learning, can be used to obtain a strategy as a POMDP path. However, the generated strategy from the virtual POMDP is not creative, since they are generated considering utility values and not novelty. To tackle the lack of evaluation with respect to novelty, it is proposed that the Bayesian surprise should be used inside an agent's reasoning within an associative phase, to generate novel actions according to external stimulus received from an environment. This method enables the simulation of a *Contextual Focus*, that swaps between being divergent or convergent, where it will be convergent when optimizing utility values and divergent when taking decisions based on surprise responses from an Expectation ART.

The proposed solution confirms a creative idea as the set of all generated actions through time, being a POMDP path or strategy, where its optimization is influenced by novel decisions triggered by a *contextual focus*. By the proposed solution, strategies will be optimized in a creative way, since novel decisions will impact in how new states, from a virtual POMDP, will be achieved. As depicted in Figure 15, the process of generating a creative decision starts by receiving an external stimulus that represents all the possible actions, a_1 to a_3, that an agent can perform in its current state and observed environment *env*. The agent will subsequently decide if it will exploit an already known strategy or explore a novel one by selecting the highest surprise action. If it determines to exploit the already known strategy, then it will pass the environment *env* as the environment field of a FALCON system, where the action a_1 is returned as a prediction, otherwise it will pass all possible actions that it can perform and the received environment *env* through a proposed Expectation ART, that calculates the Bayesian surprise of each received action, then selects the action a_3 as the one with highest surprise.

The contextual focus process, presented in Algorithm 3, works in a similar way

as an ϵ-greedy decision policy, being the variable ϵ a threshold discount factor, from the FALCON architecture, where if obeying an inequality $r < t$ it will activate the Expectation ART, otherwise it will activate the virtual POMDP FALCON.

Algorithm 3: Contextual focus as a decision policy algorithm.

Input: Environment e, Possible Actions A
Output: Predicted Action a

$s \leftarrow signal(e)$
$SA \leftarrow \emptyset$
for each $a_i \in A$ **do**
    $SA \leftarrow SA \cup signal(a_i)$
end for
$r \leftarrow random()$
if $r < t$ **then**
    $a \leftarrow POMDPANN(s)$
else
    $a \leftarrow ExpectationART(s, SA)$
end if
$t \leftarrow t - \epsilon$
return a

Source: By the Author.

7.2.3 Discussion

The proposed HARP process is based on using two measurements that will affect how to optimize the virtual POMDP. There are two primary limitations of the proposal: 1) the POMDP FALCON can behave in a way that facilitate its accelerated growth, thus resulting in a bigger searching space for the ANN five step routine; 2) it operates two neural networks in order to obtain both predictions, thus extending the total amount of memory needed to store both structures. The first problem is more concerning, since the speed in which the reasoning gives responses will impact in how often the agent can process States from a *Time Flow*. A feasible solution to this problem is proposed in Chapter 8, where fields were broken and represented as channels from a cluster of ART systems that are assembled as a hierarchical structure. By exerting this kind of structure the search space size will be reduced when performing the ANN five step routine, since neurons constitute a distributed hierarchical structure, where each layer is composed by a narrower set of neurons than a whole ANN.

8 ADAPTIVE NEURAL NETWORKS FOR CREATIVE THINKING

In this chapter are proposed three Adaptive Neural Networks to be used within The Honing Adaptive Resonance Process. The first ANN, presented in Section 8.1, is called Expectation ART and it permits to calculate the Bayesian surprise within stored expectation neurons in order to enable the HARP to complete creative behavior. On the other hand, a second proposal called Proximity Adaptive Neural Network is used to represent precise information and to allow storing compact representations of observed States to reduce the amount information stored and processed by the HARP. The last proposal, called Unstructured Areas Adaptive Neural Network, permits to represent a vast amount of information and it aims for fast recovery and prediction since is hypothesized that a creative thinking system would explore better a search space consequently demanding more storage and processing power. It accomplishes that through the concept of information sharing between neuron clusters with distinct levels of meaning.

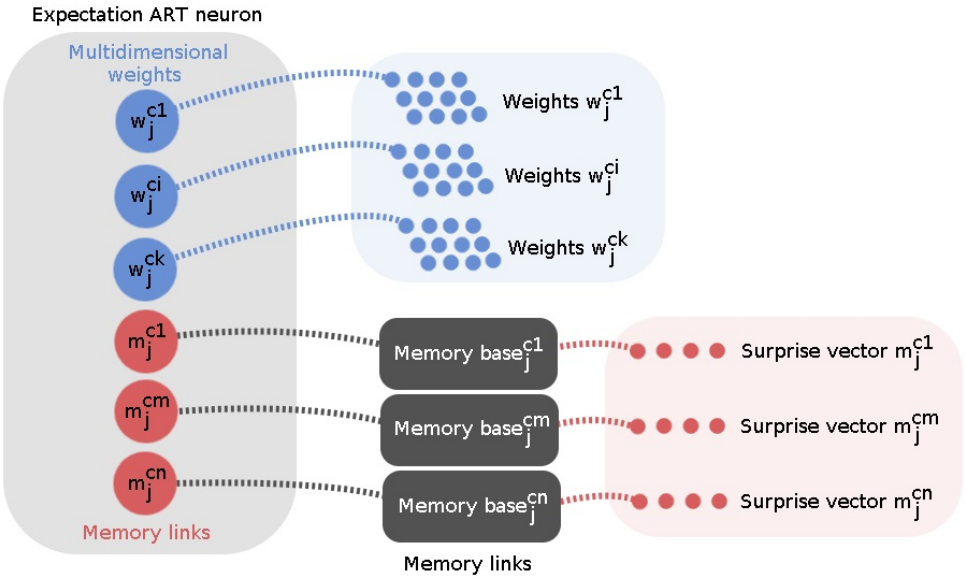
8.1 Expectation ART: Calculating the Bayesian Surprise with an Adaptive Neural Network

When calculating novelty through Bayesian surprise it is important to handle a knowledge base, used as a baseline population of already known information, where a surprise can be measured from an observation. The vital problem when dealing with such a knowledge base when controlling an agent is the fact that it performs through a POMDP, where each state maintains information about itself, exclusively, thus if trying to calculate the surprise with one global knowledge base, the results will be corrupted, because information from each state can be overlapped by others. Another problem is related to the lack of a system to store an infinite number of bases one for every viable state. To tackle these problems, it is proposed the usage of an exclusive multichannel ANN to handle Bayesian surprise knowledge bases, called Expectation ART, and it holds exclusive state expectation information about each observed environment e as surprise vectors.

The Expectation ART is defined as a multi channel ANN of two layers F_1 and F_2, as the ARAM neural network. One crucial difference is that the F_1^{ci}, representing its feature layer, is assembled as $i \in [1,\, k+n]$, where k indicates the total amount of fields used to store k environment fields information and n being the total amount of surprise

bases types linked inside a neuron. By using this F_1 layer configuration each neuron from the F_2 layer will be assembled as a set of weights W_j^{ci}, where $i \in [1, k]$, and a set of memory base links cm to surprise vectors M_j^{cm}, where $m \in [1, n]$ and j being a neuron j from W_{all} as showed in Figure 18.

Figure 18 – Expectation ART neuron with its memory links and surprise vectors.



Source: By the Author.

8.1.1 Surprise vector composition

The surprise alone can be measured with a static knowledge base, that does not change over time, or a dynamic one, that changes over time. That stems from the fact that with a symbolic system there is no necessity in updating its values since the system is all provided by a specialist. However, the expectation ART surprise should be measured with a dynamic base, since each one is created dynamically through adaptive behavior inside Expectation neurons, where the overall surprise of a previously stored vector can be obtained through Equation 25.

$$S(M_j^{cm}) = 1 - \frac{1}{1 + \sum_{i=0}^{h} \frac{N}{2\sigma_i^2}[\sigma_i^2 + (c_i^{new} - \bar{m}_i)^2]} \tag{25}$$

Equation 25 is normalized inside the interval [0, 1], where it is used inside the ART procedure for prediction, where the term $(1 + \sum_{i=0}^{h} \frac{N}{2\sigma_i^2}[\sigma_i^2 + (c_i^{new} - \bar{m}_i)^2])$ avoids it going

out of bounds.

Surprise vectors can be composed by a global or window observation. If composed by a global observation, each $M_j^{cm} = \{c_1, ...c_h\}$, where j is the current neuron, m the current field, h as the total amount of actions coded by the vector, and each $c_i \in M_j^{cm}$ being $\{\sigma, \bar{m}\, n\}$, where σ is the variance, $\bar{m}$ the mean and n the total amount of elements on the population. Each surprise vector, from a global observation, is used to compute the Bayesian surprise of an action from a starting point in time to a cumulative infinite future and it is called compact representation. The compact representation of surprise inside M_j^{cm} prevents recalculations of statistical measurements from a starting point to a cumulative infinite future, but it does not enable to represent a time window, thus it can not simulate a short-term memory for expectation. To tackle that problem, each M_j^{cm} needs to store not compact values about statistical measurements of a population, but rather the entire population measured in a time window, thus changing the form of M_j^{cm} to a dynamic population matrix Λ_j^{cm} as represented by Equation 26, where each row being equals to $\{c_1, ..., c_h\}$.

$$\Lambda_j^{cm} = \begin{pmatrix} c_{ms,1} & c_{ms,2} & \cdots & c_{ms,h} \\ c_{ms+1,1} & c_{ms+1,2} & \cdots & c_{ms+1,h} \\ \vdots & \vdots & \ddots & \vdots \\ c_{w,1} & c_{w,2} & \cdots & c_{w,h} \end{pmatrix} \tag{26}$$

Each variable $c_{ij} \in \Lambda_j^{cm}$, constitutes an action that was performed or not for the time window interval inside the range $[ms,w]$, where each row represent an environment obtained from a state of a POMDP. This coding can represent a time window, but it lacks the ability in computing surprise efficiently, thus the application nature will guide in how to represent expectation through time considering exclusive *States* information.

8.1.2 Prediction

The prediction process starts at step one, from the multichannel ANN, where neurons compete against each other through a measured temperature by an activation function in Equation 14. All neurons, from activation, will be checked one by one, through the inhibition process by Equation 4, then, resonance is checked to guarantee the quality of the prediction through Equation 7. The most promising neuron, as the resonating one

with the most elevated temperature, is used to extract the surprise of each action that will be used for readout. The surprise is calculated by vector types linked by all *Memory bases* inside an Expectation ART neuron, where each $F_1^{ci}|i \in [k+1,\, n]$, will be used alongside its counterpart $M_j^{cm} \in F2$ in Equation 25 and stored inside $S = \{s_{a1}, ..., s_{an}\}$, where n is the total number of valid actions. The most promising action will attend the one with the highest surprise selected by Equation 27, where it is used to consider a novel decision.

$$s_A = \max\{s_{ai} : \text{for all } a_{ai} \in S\} \tag{27}$$

A direct readout occurs through an activity vector V in F_1^{ci}, where $v_i \in V = 1$ if the action is selected or $v_i \in V = 0$ if the action was unselected. Furthermore, a indirect readout can also occur by copying each obtained surprise for actions inside A, from Equation 25, into V, where $v_i \leftarrow a_{ai}$ for each $a_{ai} \in A$.

8.1.3 Learning

The learning process occurs as in the original fuzzy multichannel ANN, however the surprise bases for the selected learning neuron J, from the ANN learning steps, need to be updated to obtain an adaptive behavior over time through the presentation of various environments. The essential problem is the lack of a way in storing σ and $\bar{m}$ of a population to fast recalculate a new σ_{new} and $\bar{m}_{new}$ based on what action was selected through the prediction process. To tackle this problem, each variable, σ, $\bar{m}$ and n inside c_i, is used to store previously calculated statistical measurements from a population that a surprise vector represents, and consequently updating its values based on Equation 28.

$$c_i = \frac{\sigma_{ci} n_{ci} + (v_i - \bar{m}_{ci})^2}{n_{ci}}, \frac{\bar{m}_{ci} n_{ci} + v_i}{n_{ci}}, n_{ci} = n_{ci} + 1 \text{ for each } c_i \in M_j^{cm} \tag{28}$$

This learning model makes possible to comprise a global population of what an agent had performed in a POMDP. When using a window coding, Equations 29 and 30 should be used instead for learning, where $c_{i,j} \in \Lambda_j^{cm}$ is an element from the row i and column j, and $c_{w,j}$ the element from the j_{nth} column from line w.

$$c_{i,j} = c_{i+1,j} \tag{29}$$

$$c_{w,j} = v_i \text{ for each } v_i \in V|i = j \tag{30}$$

8.2 Proximity Adaptive Neural Network for Precise Matching

The ANN operates with ART I or ART II as composite operations within its working mechanism to categorize neurons in a stable and fast way. The problem of most concern is the lack of the system in dealing with precise categorizations, thus turning unfeasible the use of other types of representations inside each input vector. Even though, categorizing with a high resonance value for ρ, can be unguaranteed that the neuron matching will occur correctly for prediction and learning. Besides that, there is the possibility of a accelerated growth of input vectors to constitute actions, since the ANN uses one position i for each action inside its fields, increasing the computational cost of the algorithm. This chapter presents a solution for those problems in the form of a new ANN that operates with Manhattan distances for its *activation*, *inhibition*, *learning* and *prediction* steps. Furthermore, to solve the lack of precision is proposed that *resonance checking* should be done also for the *prediction* step. Besides that, it is also proposed that a *Spectrum* model, that demands a precise categorization mechanism, for action coding should be used for substantial amounts of actions.

8.2.1 Spectrum coding

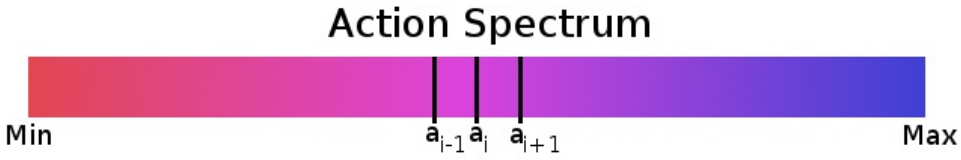
The categorization mechanism of the five step routine of the ANN works with a feature vector of the form $P_i^1 = \{p_1, ..., p_n\}$, where each $p_i \in P_i^1$ is a value inside the interval $[0, 1]$. There is one problem with this coding refereed as the lack of the system in dealing with compact representations. For example, if the maximum number of actions that an agent can perform is equal to 10, then the action field P_2^1, responsible for holding all actions, will codify each one of them as a binary variable, thus creating one item per action inside P_2^1 creating an unnecessary data overhead. A valid solution would be using binary coding instead, however, the network can not code any stimulus in a precise way as argued in Section 4, thus, even coding with binaries, the network will not categorize them correctly influencing an agent to achieve incorrect actions. Even when using binary coding, an unnecessary overhead can be caused by the number of bits used to code all actions that the agent can perform and also by the complement coding. To tackle the overhead problem, in this research is proposed that a *Spectrum* model should be used instead.

In the ANN from Chapter 4, if an agent has a thousand actions, then according to the FALCON model it should code one action per variable inside its action field. This

type of coding facilitates using actions as an action mask, where each position i, from the action field, represents an usage ratio for the action i. This model of coding actions seems suitable if an agent considers fewer actions that it can perform, however if its action model is larger, a compact model should be appropriate. The *Spectrum* model is defined as a value from a range $R = [min, max]$ that can vary, where min is the min and max the max values of the range. This model is effective to code all actions by dividing an action $Identificator \geq 0$ by the interval upper bound max, where $max \leq 1$ and $min \geq 0$. For example if an agent has 1000 actions, them the action 1 will be represented as 1/1000 or 0.001, thus instead of creating a vector of a thousand variables all actions can be coded with one.

This action coding scheme is represented by the restriction $a_i \neq a_{i+1} \forall a_i \in A$, where each action $a_i \in A$ has a different and unique position inside the spectrum, for all possible actions a_{i-1} and a_{i+1}, as depicted in Figure 19.

Figure 19 – Action spectrum coding model.



Source: By the Author.

Coding actions with the proposed spectrum model for an ANN will cause a drastically reduction on the size of the action input vector from the action field on a FALCON, thus coding them also as composite actions of the form $P_2^1 = \{P_1, ...P_m\}$, being each $P_i \in P_2^1$ action set. This model permits the composition of actions described by more than one variable, for example, if an agent can complete actions from two sets $P_1 = \{a_1, a_2\}$ and $P_2 = \{t_1, t_2\}$, where a_1, a_2 are action names, t_1 and t_2 are types, them it can be represented as $P_2^1 = \{P_1, P_2\}$. Its feature vector would be $c = \{\frac{ID_1}{2}, \frac{ID_2}{2}\}$, where ID_1 and ID_2 represents the action position inside each vector. This method is useful if it is essential to separate actions by type, class or even range, since, with the spectrum model, every possible action can be coded with one variable that will indicate its position.

8.2.2 Proximity Based Categorization for an Adaptive Neural Network

To solve the problems presented in Section 4, this research proposes the utilization of the normalized sum of the individual euclidean distances, presented by Equation 31, to calculate the temperature or similarity between the stimulus x^{ci} and the neuron weight w_j^{ci}.

$$t_j \in T = \frac{\sum_{i=1}^{k} \frac{1}{n^{ci}} \sum_{z=1}^{n^{ci}} \sqrt{(x_z^{ci} - w_{jz}^{ci})^2}}{m} \tag{31}$$

This metric can seem as more precise than both ART I and ART II since ART I use the fuzzy and operation as the minimum between the stimulus and neuron weights it can generalize but never specialize. Also, if analyzed, the ART I operation considers the length relation between input and codified neuron weights, thus leading to categorizations of divergent classes inside the same neuron. The ART I operation is efficient to generalize as addressed by (CARPENTER; GROSSBERG; REYNOLDS, 1991; TAN, 1995; WANG; TAN, 2015) but the over generalization causes data corruption after a few training steps. The fuzzy ART II was created to solve those problems, as a precise metric, but it groups received stimulus by direction. Since ART II calculates the cosine of the angle between the received stimulus and neuron weights, thus stimulus with distinct characteristics, that does not appear to be similar, can be falsely categorized in the same category.

A further simplification of the Equation 31 could be done, because when $\sqrt{(x_z^{ci} - w_{jz}^{ci})^2}$ are added separately, then it turns into $|x_z^{ci} - w_{jz}^{ci}|$ that is equal to the Manhattan distance. For easy understanding, the Equation 31 can be divided into two parts presented by Equation 32 and 33, as a Manhattan distance metric.

$$norm_d(i,j) = \frac{\sum_{z=1}^{n^{ci}} |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}} \tag{32}$$

The $norm_d(i,j)$ is the normalized Manhattan distance between the received stimulus x^{ci} and the neuron weight w_j^{ci} from the field ci, and n^{ci} the total number of components or the length of x^{ci} and w_j^{ci}. This distance calculates the level of similarity without despising individual components of the received stimulus and neurons weights, thus it guarantees that not the length is considered when categorizing. The normalization of $norm_d(i,j)$ by n^{ci} is important to maintain the consistency when operating inside the network, and by

doing this the similarity will always be inside the interval [0, 1], and can be used for other purposes as a normalized metric.

$$t_j = \frac{\sum_{i=1}^{k} norm_d(i,j)}{m} \tag{33}$$

The global similarity value t_j for the neuron j is computed by the sum of all the normalized Manhattan distances from all fields ci for $c = 1, ..., k$. The value of t_j is normalized by m, the total of fields inside the multi-channel ANN, to use it as a normalized metric within the proposed inhibition method described in Section 8.2.3. This value represents the total normalized temperature or similarity between the received stimulus X and W_j, and it ranges inside the interval [0, 1]. The closeness of t_j to 0 implies into a nearly perfect match between the verified stimulus and the neuron, otherwise, if t_j gets close to 1 this implies in a completely different stimulus.

After the calculation of T and its values, the readout can not occur as in the common multi-channel ANN, the proposed solution needs to calculate the resonance first and thus ensure the satisfaction of all the vigilance inequalities to obtain a balanced match between neuron and stimulus. To achieve this, Equation 32 was used within the resonance checking procedure for each field ci for all $i = 1, ..., k$. The new inequality for resonance checking, created based on Equation 32, is presented by Equation 34.

$$m_j^{ci} = 1 - norm_d(i,j) > \rho^{ci} \tag{34}$$

The $norm_d$ was subtracted from 1 permitting ρ^{ci} to represent the percentage of matching needed in the field c^{ci} to get a resonance as in the original multi-channel ANN. It is also important to note that the calculation of the new inequality is done at the same moment when calculating Equation 32, thus saving a lot of processing time. If the results of the inequality in the Equation 34 returns true, then m_j^{ci} is set to true. If resonance occurs when $m_j^{ci} = true$ for all $i = 1, ..., k$, then y_j is set to an arbitrary value higher than a predefined constant $\zeta > 1$, otherwise $y_j = 0$. This value needs to be higher than ζ to conform with the neuron inhibition method presented in the Section 8.2.3.

8.2.3 Inhibition method

An inhibition process is what gives the network the capacity to choose appropriate neurons for the execution of the learning and readout operations. In the approach presented

in this research, the inhibitory process is accomplished by getting the max neuron activity $a_j \in A$, as presented by the Equation 35.

$$a_J = \max\{a_j : \text{for all } a_j \in A\} \tag{35}$$

The difference from Equation 4 is that $a_j = y_j + (1 - t_j)$ for all $a_j \in A$. This puts each a_j inside the range $[0,\ 1 + y_j]$, where each a_j is the sum of the inverse normalized Manhattan metric plus the resonating value represented by $y_j > 1$. A resonated neuron can be surely selected after the calculation of the Y vector, but this will also output an overgeneralized response from the network because checking if a neuron resonated or not, does not guarantee the real similarity between the received stimulus and the neuron weights. The reciprocal cannot be done either as in the original multi-channel ANN, because selecting by the similarity does not guarantee the neuron is in balance with the received stimulus. By saying that, the most promising or the most reliable neuron J is the one with the higher similarity among all the resonated neurons, that can now be extracted directly with the max operation from the newly computed vector A. If a_j is a non-committed neuron, then $a_j = \lambda$, where $1 < \lambda < y_j$ to ensures the selection of a non-committed neuron if none of the $y_j \in Y$ resonate. It is important to note that a_j must be equal to λ if the network is operating in learning mode.

8.2.4 Prediction and learning

A stimulus prediction for an arbitrary field ci can be obtained by multiplying Equations 32 and 34 by an $\gamma^{ci} = 0$, where $\gamma \in [0,\ 1]$, thus generating Equations 36 and 37.

$$norm_d(i,j) = \gamma^{ci} \frac{\sum_{z=1}^{n^{ci}} |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}} \tag{36}$$

$$m_j^{ci} = (1 - norm_d(i,j)) > \gamma^{ci}\rho^{ci} \tag{37}$$

This procedure will force the network to ignore the field ci, ensuring that it will not generate influence in the cognitive code matching neither in the inhibition procedure. To control individual components influence directly inside x^{ci}, the mask G can be used. Thus $|x_z^{ci} - w_{jz}^{ci}|$ is multiplied by $g_z \in G$, for fine control over each variable influence inside

the cognitive code matching. This form of coding the neuron similarities is represented by the Equation 38.

$$norm_d(i,j) = \gamma^{ci} \frac{\sum_{z=1}^{n^{ci}} g_z |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}} \tag{38}$$

For a prediction to be achieved, the full cognitive code matching and inhibition method must be executed. The cognitive code matching could follow either Equation 36 for common prediction or Equation 38 for a finetuning prediction, and the resonance vector Y must be calculated through Equation 37. The process of prediction ends when the readout of the selected neuron J is performed with Equation 6 to preserve the integrity of the received stimulus presented in x^{ci}. The full process terminates when learning if the systems are in learning mode the received stimulus is learned by using Equation 9 to preserve the integrity of the received stimulus.

8.3 Unstructured Area Multi-channel Adaptive Neural Network: Representing Vast Amounts of Information

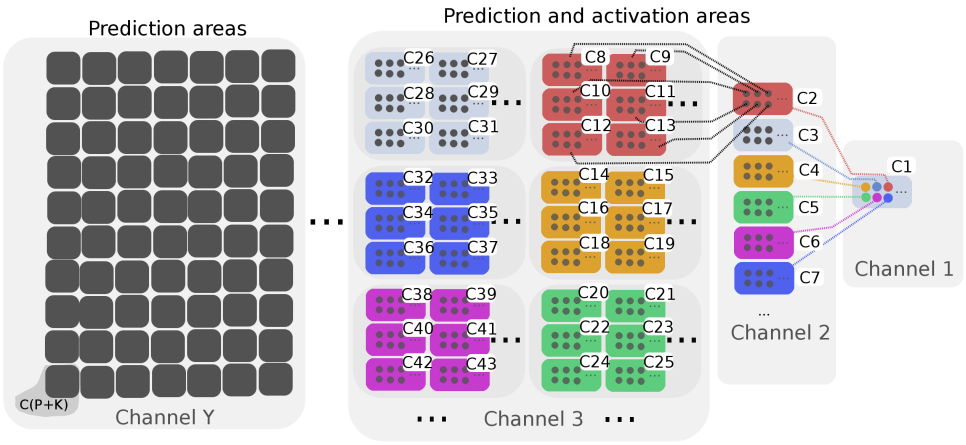
In order to represent semantic information in a more simplistic way, the proposed UAM-ANN organizes ART fields into input channels, where each input channel is composed of isolated neuron clusters called areas. Channels are organized in a hierarchic way, from top to bottom, where the top channels represent lower level information and higher channels represent high level information. Lower levels are used to represent memories obtained from the ART. On the other and higher levels are used to represent *Semantic Objects*. Higher channels can be seen as the combination of lower channels to represent semantic information. By using this structure, the network can handle different neuron types for each channel, what permits modularity, and go beyond by reducing the search space when seeking for a prediction.

8.3.1 Channel structure

The essential structure of the proposed UAM-ANN (Unstructured Area Multi-channel Adaptive Neural Network) is defined as a set $C = \{c_1, ..., c_{k+p}\}$, where each $c_i \in C$ represents a neuron cluster, made of ART neurons, called area. The size of C can vary according to the networks interaction with an environment, thus all available areas are represented by $k + p$, where k represents activation areas and p prediction areas. Each area represents a multi-channel ANN with two layer, $F1$ and $F2$. The $F1$ layer is used to activate neurons and $F2$ used to store all neurons from that area. Areas are organized within channels, where each $c_i \in C$ belongs to one, and one, channel ch_i from the channel set CH. Each channel has an input activity denoted as $X^{ch_i} = \{X_1^{ch_i}, ..., X_f^{ch_i}\}$ and an output activity denoted as $O^{ch_i} = \{O_1^{ch_i}, ..., O_f^{ch_i}\}$, where each $X_h^{ch_i} \in X^{ch_i} = \{x_{h,1}^{ch_i}, ..., x_{h,n_h^{ch_r}}^{ch_i}\}$ and each $O_h^{ch_i} \in O^{ch_i} = \{o_{h,1}^{ch_i}, ..., o_{h,n_h^{ch_r}}^{ch_i}\}$, being h a field, p a variable at the $p$nth position and $n_h^{ch_r}$ the total number of variables for the field h. The example illustrated in Figure 20 represents an unstructured area multi-channel ANN, where the area c_1 belongs to channel 1 and it is composed by 6 or more ART neurons, where each one of them is linking an area in channel 2 from c_2 to c_7 to c_i. The last channel is called Y and it represents a final prediction channel.

Areas can be of two types: 1) Prediction; 2) Activation. Prediction areas are composed by more than one field, thus it is possible to obtain a prediction from them.

Figure 20 – Unstructured area multi-channel Adaptive Neural Network scheme.



Source: By the Author.

Moreover, activation areas are made by one field, thus they are used to activate other areas. Each neuron from an area in channel ch_i have a link to an area in channel $ch_{(i+1)}$, thus channels are indirectly connected in a hierarchic way through prediction and activation areas. In order to facilitate communication and manipulation of areas, the input and output activity vectors for a channel are plugged to all $F1$ layers from all areas within that channel to send and extract signals from them.

8.3.2 Activation area

Each $c_i \in C = \{F1^{ci}, F2^{ci}\}$ in channel ch_i, where $F1^{ci}$ is the feature layer and $F2^{ci}$ the category layer for the activation area i. The $F1^{ci}$ is responsible for holding the activation vector X^{ci} for the area i, where $X^{ci} = \{x_1^{ci}, ..., x_n^{ci}\}$, being each $x_i^{ci} \in X^{ci}$ a variable used to represent a received external stimulus and n the total number of variables. The category layer, differently, is composed of $F2^{ci} = \{N_1^{ci}, ..., N_t^{ci}\}$, where t is the total number of neurons and each $N_i^{ci} \in F2$ is an ART neuron from the ci area. Each neuron $N_j^{ci} \in F2^{ci}$ is composed as $\{w_{j,1}^{ci}, ..., w_{j,n^{ci}}^{ci}, c(ij)\}$, where each $w_{j,i}^{ci} \in N_j^{ci} \in [0, 1]$, $c(ij)$ being a activation area linked by N_j^{ci} to an area in channel ch_{i+1} and n^{ci} the total number of variables used to code N_j^{ci}.

8.3.3 Prediction area

A prediction area predicts information based on an incomplete observation from an environment e. It is defined as $c_{ip} \in C$ in channel ch_i, where each $c_{ip} = \{F1^{cip}, F2^{cip}\}$,

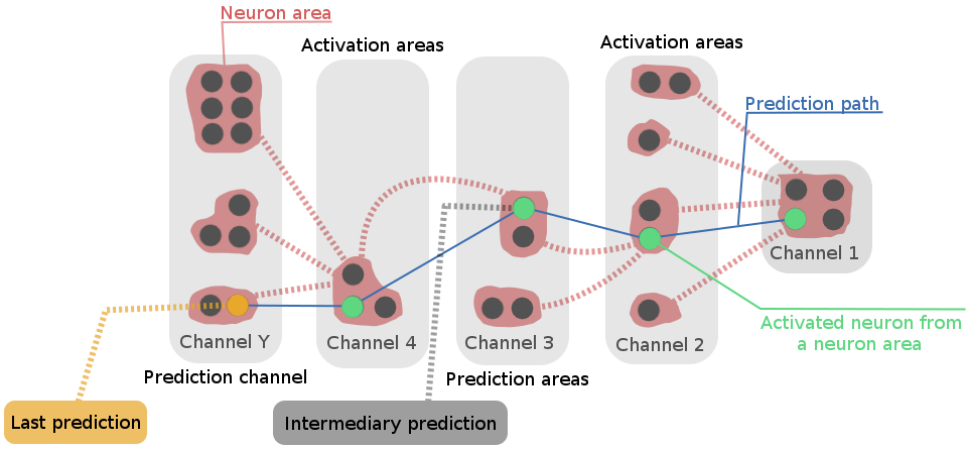
being $F1^{cip}$ a feature layer and $F2^{cip}$ a category layer. The $F1^{cip}$ layer is composed by the activity vectors $X^{cip} = \{X_1^{cip}, ..., X_f^{cip}\}$, where each $X_i^{cip} \in X^{cip} = \{x_{i,1}^{cip}, ..., x_{i,n_i^{cip}}^{cip}\}$ and f the total number of fields used to code the activity vectors. Each $X_i^{cip} \in X^{cip}$ is used to represent the field i inside from the cip area, each $x_{i,j}^{cip} \in X_i^{cip}$ is the variable value at the j_{nth} position and n_i^{cip} is used to indicate the total number of variables from field i. Moreover, the $F2^{cip}$ is described as a set $\{N_1^{cip}, ..., N_z^{cip}\}$, where z is the current number of neurons coded by the area cip. Each $N_j^{cip} \in F2^{cip}$ is an ART neuron composed as $\{W_{1,j}^{cip}, ..., W_{f,j}^{cip}\}$, where $W_{i,j}^{cip} \in N_j^{cip}$ is equals to $\{w_{i,j,1}^{cip}, ..., w_{i,j,n_i^{cip}}^{cip}, c(ij)\}$, being i the $i$nth field from $F1^{cip}$, j the j_{nth} neuron from $F2^{cip}$, $w_{i,j,p}^{cip}$ the variable at the $p$nth position in $W_{i,j}^{cip}$, n_i^{cip} the total number of variables for the $i$nth field and $c(ij)$ a link to an area in channel ch_{i+1}.

8.3.4 Prediction

The prediction mechanism for the proposed network also works with a routine described as: 1) Activating; 2) Inhibition; 3) Resonance checking; 4) Reset; and 5) Readout. It starts by receiving an external stimulus in all channels simultaneously and storing their input signal into their X^{ch_i} activity vectors. At this stage, the network activates the first channel and transfers the input from X^{ch_1} into the $F1$ layer from area $C1$. With that data stored there, the network then completes the activation, where neurons from $C1$ compete against each other in order to be selected. For instance, the example in Figure 21 shows an unstructured area multi-channel ANN with 4 channels, where two of them are prediction channels. In the shown example, the prediction process starts by activating all neurons from area $C1$ in channel 1 upon the presentation of an external stimulus to the $F1^{ch_1}$ layer. The activated neurons pass through an inhibition process and resonance checking process in order to select the most compatible neuron with the received external stimulus. If the selected neuron does not pass the resonance checking, then a reset occurs and a new neuron will be searched from that area. The selected neuron, represented by green nodes, activate an area in channel 2, thus reducing the search space by excluding neurons that are unrelated to the initial received external stimulus in channel 1.

When reaching channel 2, a new input needs to be presented in order to perform another activation and resonance checking process enable to achieve higher channels. At each inhibition, selected neurons can predict within their fields if the channel is assembled

Figure 21 – Unstructured area multi-channel Adaptive Neural Network area activation example.



Source: By the Author.

with Prediction areas, as shown by the *Intermediary prediction* node in gray. When reaching the last channel, called channel Y, the last prediction will be achieved as illustrated by the yellow node. The set of all predictions forms a full memory, represented by the blue path, considering the presentation of external stimulus at each channel. This model does not reduce the searching space but constitutes fields with unstructured information, thus being more flexible than the original ANN where each neuron has one static field structure.

8.3.5 Neuron Activation functions

An activation process can be accomplished by three types of coding, ART I, ART II or Proximity. In order to generalize larger amounts of information, neuron activation should be accomplished by ART I operations described by Equations 39, for Prediction areas, and 40, for activation areas. Those equations calculate the normalized sum of the input length compared to the stored weights inside all neurons to verify the similarity level between them.

$$t_j^{ch_r,cip} = \sum_{h=1}^{f} \gamma_h^{ch_r} \frac{\sum_{p=1}^{n_h^{cip}} x_{h,p}^{ch_r} \wedge w_{h,j,p}^{cip}}{\alpha^{ch_r} + \sum_{p=1}^{n_h^{cip}} w_{h,j,p}^{cip}} \tag{39}$$

$$t_j^{ch_r,ci} = \frac{\sum_{p=1}^{n^{ci}} x_p^{ch_r} \wedge w_{j,p}^{ci}}{\alpha^{ch_r} + \sum_{p=1}^{n^{ci}} w_{j,p}^{ci}} \tag{40}$$

The operator $\wedge$ is the fuzzy min, $\gamma_h^{ch_r}$ is the channel influence on the activation from neuron j, $x_{h,p}^{ch_r}$ the activity vector variable p in X^{ch_r} from h field, $w_{h,j,p}^{cip}$ the neuron

weight for variable p, f the total number of fields, n_h^{cip} the total number of variables for the field h, and α^{ch_r} a parameter from channel ch_r to prevent divisions by zero.

In order to generalize and permit some degree of specialization inside neurons, the ART II operations described by Equations 41, for Prediction areas, and 42, for Activation areas should be used. Those metrics calculate the angle between a received stimulus from a channel ch_r and neuron weights for all $w_{h,j,p}^{cip}$.

$$t_j^{ch_r,cip} = \sum_{h=1}^{f} \gamma_h^{ch_r} \frac{\sum_{p=1}^{n_h^{cip}} x_{h,p}^{ch_r} \times w_{h,j,p}^{cip}}{\sum_{p=1}^{n_h^{cip}} (x_{h,p}^{cip})^2 \times \sum_{p=1}^{n_h^{cip}} (w_{h,j,p}^{cip})^2} \tag{41}$$

$$t_j^{ch_r,ci} = \frac{\sum_{p=1}^{n^{ci}} x_p^{ch_r} \times w_{j,p}^{ci}}{\sum_{p=1}^{n^{ci}} (x_p^{ch_r})^2 \times \sum_{p=1}^{n^{ci}} (w_{j,p}^{ci})^2} \tag{42}$$

Precise categorization can be achieved with the proposed metrics from Chapter 8. They are described by Equations 43, for Prediction areas, and 44, for Activation areas. They calculate the normalized distance between the received stimulus in each $x_{h,p}^{ch_r}$ and $w_{h,j,p}^{cip}$ is calculated through an euclidean method, being m^{ch_r} the total number of variables from a field h in channel ch_r used to normalize the activation between [0,1].

$$t_j^{ch_r,cip} = \frac{\sum_{h=1}^{f} \frac{1}{n_h^{cip}} \sum_{p=1}^{n_h^{cip}} g_p^{ch_r} \times \sqrt{(x_{h,p}^{ch_r} - w_{h,j,p}^{cip})^2}}{m^{ch_r}} \tag{43}$$

$$t_j^{ch_r,ci} = \frac{1}{n^{ci}} \sum_{p=1}^{n^{ci}} g_p^{ch_r} \times \sqrt{(x_p^{ch_r} - w_{j,p}^{ci})^2} \tag{44}$$

Finally, a simplified activation with a proximity metric, Equations 45, for Prediction areas, and 46, for Activation areas, should be used since they calculate a Manhattan distance normalized metric between a received activity in each $x_{h,p}^{ch_r}$ and each neuron weight $w_{h,j,p}^{cip}$.

$$t_j^{ch_r,cip} = \frac{\sum_{h=1}^{f} \frac{1}{n^{cip}} \sum_{p=1}^{n_h^{cip}} g_p^{ch_r} \times |x_{h,p}^{ch_r} - w_{h,j,p}^{cip}|}{m^{ch_r}} \tag{45}$$

$$t_j^{ch_r,ci} = \frac{1}{n_h^{ci}} \sum_{p=1}^{n^{ci}} g_p^{ch_r} \times |x_p^{ch_r} - w_{j,p}^{ci}| \tag{46}$$

8.3.6 Area inhibition and retrieval

The inhibition process is performed after activation and it calculates which neuron will be selected after activating all of them from an area $c(ij)$ from a channel ch_r. The selected neuron J is accomplished by following Equation 47.

$$t_J^{ch_r,cip} = \max\{t_j^{ch_r,cip} : \text{for all } t_j^{ch_r,cip} \in T^{ch_r,cip}\} \tag{47}$$

Where $T^{ch_r,cip}$ is holding all neuron activations for the ch_r channel and $c(ij)$ area. The retrieved area c_{ir} is selected from the activated neuron $t_J^{ch_r,cip}$ weights from its c_{r+1} variable that holds the area link from the ch_{r+1} channel. The selected area is also returned into a new activation and inhibition process, thus permitting it to reiterate until it is performed on an area from the last channel Y in order to achieve a decisive response.

8.3.7 Learning

The learning process creates areas and learns previously received stimulus. It is proposed in this solution that the learning process should occur in two stages, where in stage 1 new areas can be created and in the stage 2 neurons can adjust its weights according to the received external stimulus from each X_{ch_i}.

Stage 1 learning: This stage occurs when activating neurons from any channel ch_r different from channel Y, where a new neuron $N_j^{ch_r,c_{ij}}$, from area $c(ij)$, is created if it does not exist. In this case, neuron weights are updated with Equation 48.

$$w_{h,j,p}^{c(ij)} = x_{h,p}^{ch_r} \text{ for each } w_{h,j,p}^{c(ij)} \in N_j^{ch_r,c(ij)} \tag{48}$$

Where h is the current field, j a newly created j_{nth} neuron, p the variable a the $p$nth position inside the h field and $x_{h,p}^{ch_r}$ the activity inside the $F1^{c_{ij}}$ layer from area $c(ij)$ inside a channel ch_r. This learning should occur for each present field inside $N_j^{ch_r,c_{ij}}$. The created neuron will finally link a new area c_{ik} inside its weights, turning possible for the activation procedure to go through all channels until reaching a channel Y.

Stage 2 learning: This stage occurs if the selected neuron passes through a resonance checking, thus being enabled for learning, or a **Stage 1 learning**. It performs weights adjustments with ART I or ART II learning methods. The ART I learning described

by Equation 49, adjust weights according to the fuzzy AND operation, thus it generalizes an amount of the received stimulus in each $x_{h,p}^{ch_r}$. If following the ART II method, neurons weights are accomplished with Equation 48.

$$w_{h,j,p}^{c(ij)} = w_{h,j,p}^{c(ij)} \wedge x_{h,p}^{ch_r} \text{ for each } w_{h,j,p}^{c(ij)} \in N_j^{ch_r,c(ij)} \tag{49}$$

Where h is the current field, j an already created j_{nth} neuron, p the variable a the $p$nth position inside the h field and $x_{h,p}^{ch_r}$ the activity inside the $F1^{c_{ij}}$ layer from area $c(ij)$ inside a channel ch_r.

8.3.8 Resonance checking and reset

A resonance checking can be performed on top of a selected neuron $t_J^{ch_r,cip}$, from the activation and inhibition processes, in order to ensure certain level of matching according to a resonance checking parameter p^{ch_r}, thus selecting reliable neurons for learning or predicting. It can be accomplished by the proposed proximity metric, with Equations 50 and 51, or with Equation 52.

$$norm_d(h,J) = \frac{\gamma_h^{ch_r} \sum_{p=1}^{n_h^{c(ij)}} |x_h^{ch_r} - w_{h,J,p}^{c(ij)}|}{n_h^{c(ij)}} \tag{50}$$

$$m_{h,J}^{ch_r,c(ij)} = (1 - norm_d(h,J)) > \gamma_h^{ch_r} \rho_h^{ch_r} \tag{51}$$

$$m_{h,J}^{ch_r,c(ij)} = \frac{\sum_{p=1}^{n_h^{c(ij)}} x_{h,p}^{ch_r} \wedge w_{h,j,p}^{c(ij)}}{\sum_{p=1}^{n_h^{c(ij)}} x_{h,p}^{ch_r}} > \rho_h^{ch_r} \tag{52}$$

Where ch_r is the current channel, $c(ij)$ the area where resonance is being checked, J the selected neuron from the inhibition process, p the current variable inside neuron weights, h the current field, if performing in a Prediction area, and $n_h^{c_{ij}}$ the total number of variables on the h field. In order to resonate, the neuron $m_{h,J}^{ch_r,c_{ij}}$ needs to obey the resonance checking rule for all fields in ch_r channel, otherwise a reset occurs and a new activation and inhibition processes will select a new neuron to be checked. If no resonance occurs after checking all neurons in $c(ij)$, then a new neuron is created in $c(ij)$ with **Stage 1 learning**.

8.3.9 Readout

A readout operation can be performed within Prediction areas and is accomplished by a direct access method or by inhibiting the $\rho_h^{ch_r}$ for the field that will be predicted from area *cip* in channel chr_r. The direct access method works by setting $x_{h,p}^{ch_r} = 1$ for each $x_{h,p}^{ch_r} \in X_h^{ch_r}$ in channel ch_r and field h. This will permit the activation and inhibition processes in ignoring the h field when performing, thus all ignored fields will perform a readout after selecting the J_{nth} neuron. The $\rho_h^{ch_r}$ for the h field works with the proximity activation metrics, where the h field will be suppressed during activation and inhibition processes. The readout can follow Equation 53, in order to obtain an ART I response from the network, or Equation 54, in order to obtain an ART II response accomplishing a more precise response if needed.

$$o_{h,p}^{ch_r} = w_{h,J,p}^{c(ij)} \wedge x_{h,p}^{ch_r} \text{ for each } w_{h,J,p} \in W_{h_J}^{c(ij)} \tag{53}$$

$$o_{h,p}^{ch_r} = w_{h,J,p}^{c(ij)} \text{ for each } w_{h,J,p} \in W_{h_J}^{c(ij)} \tag{54}$$

Where h is the field that will be given as a response by the readout, J the selected neuron from the inhibition process, $c(ij)$ the working field inside ch_r, $w_{h,J,p}^{c(ij)}$ a variable from the selected neuron J weights $W_{h,j}^{c(ij)}$. This type of readout will perform a generalization in all responses, thus it need to be used with caution.

Part III

Deploying the Honing Adaptive Resonance Process in HearthStone agents

9 HEARTHSTONE MODELS FOR AN ADAPTIVE NEURAL NETWORK

In this chapter is presented all the proposed *Hearthstone* feature vector compositions in order to represent a game State from its *Time Flow*. The proposed model assumes that high level information is already available from the game and coded into symbols. Four models are proposed in order to represent the game, where two of them are referring to the game environment, what a player can see from a State, and the other two that refers how to represent actions that an agent can take.

In the digital collectible card games universe, *Hearthstone* is one of the most played games in the last year (BLIZZARD, 2018; GóES et al., 2016). The game mechanics rely on alternating turn matches between two players, where each player tries to destroy a controllable avatar called hero from the opposite player. If the hero avatar of one player is destroyed, then the opposite player wins the game. Each avatar has health points limited to a maximum of thirty points, and each avatar also have extra health points stored as armor points. There are currently nine available hero avatars to play with: 1) Druid; 2) Hunter; 3) Mage; 4) Paladin; 5) Priest; 6) Rogue; 7) Shaman; 8) Warlock; 9) Warrior. A player also has resources called *mana* crystals and a battlefield depicted in Figure 22.

Figure 22 – *Hearthstone* battlefield from Blizzard's Entertainment all rights reserved (BLIZZARD, 2018).

Source: By the Author.

On each turn, a player draws a card from his 30-card deck and play cards from his hand, summoning minions to the battlefield or/and casting spells on characters. The

battlefield is where most of the combat actually happens. It can accommodate up to 7 minions on each player's side at the same time. Minions that are not destroyed in the current turn, usually stay in the battlefield for the next turn.

Before engaging in a match, a player has to pick a hero that represents a particular class, with specific cards and hero power. Then he has to build a deck of 30 cards composed of the specific hero's cards but also of neutral cards, that are common to all classes. There are mainly two types of cards: minion and spell. Each minion card commonly has attributes such as attack points, health points, mana cost and effects. The attribute *attack points* indicates how much damage a minion can inflict, and *health points* how much damage it can take before it is destroyed. The *mana cost* attribute dictates how much resource (mana crystals) a specific card costs to be played. Finally, the attribute *effects* can vary from increasing the damage of a minion, restoring hero's health points to freezing an enemy minion and many others. On the other hand, spell cards have *mana cost* and *effect*.

Once a game match starts, each hero begins with 30 health points and is defeated when its health points down to zero. On each turn, a player can play any cards from his hand, use his hero power or minions to attack characters (minions or hero) and particularly combining cards, that is, playing *combos*. In this research, a combo is defined as a group of related cards played in the same turn, independently of order.

9.1 Hearthstone search space size assumptions

In order to estimate how many POMDP States an agent needs to deal with in *Hearthstone*, some assumptions were made: 1) each player can have a maximum of 30 health points; 2) 2 copies of the same card in the deck; 3) can not steal cards from enemy's deck; 4) can not duplicate cards from their decks; 5) each card have one instance, where cards affected by buff effects, that changes their stats, will not be considered new instances; 6) and a player has 100 hero cards plus 717 neutral cards to build his own deck, thus giving him 817 cards to play with.

Considering all assumptions, an estimation on how many battlefields configurations *Hearthstone* can have for one player for all nine heroes and all battlefield sizes is given as $\sim 1.78 \times 10^{23}$. The proposed estimation for the total amount of states considering one player is equals to $1.78 \times 10^{23} \times health \times mana$ and for two players it is equals to $\left(1.78 \times 10^{23}\right)^2 \times health^2 \times mana^2 \approx 2.85 \times 10^{51}$, where $health$ is the assumed total amount

of health, and *mana* the assumed total amount of mana. It is important to note that this estimation counts the quantity of states in a POMDP and not paths that can be formed by combining them.

For all the assumed conditions, it is estimated that *Hearthstone* has 2.85×10^{51} possible states. However, when computing paths between them, the result shows a number equals to $\sum_{p=0}^{m}(2.85 \times 10^{51})^p$, where p is the size of the path and m the maximum allowed path size. This estimation can not be calculated easily since $m = \infty$ if the rule that estimate the game time limit is abdicated. If estimating for paths where $m = 2$ the results leads to $\sim 8.12 \times 10^{102}$. This estimation is absurdly higher than 10^{81} that represents the total estimated amount of fundamental particles on the visible universe (POUNDSTONE, 2013). The nature of *Heartstone* provides a search space so big that is practically impossible to generate strategies by hand or without some sort of advanced cognition. Reasoning in HearthStone is a hard task to achieve, since its search space size when estimated only for $m = 1$ is more than half of the total amount of estimated state, equal to 250^{150}, for the GO game (SILVER et al., 2016).

9.2 Micro and macro models

In this research, it is proposed that the *Hearthstone* model should be accomplished according to a granularity level. This granularity level imparts how much information is used to describe game objects and is represented by two models: *Macro* and *micro*. The example illustrated in Figure 23 shows a *micro* model for the card *Wildhammer Keeper*, where the card characteristics are used to describe the card with two symbols: *Taunt* and *Overload*.

Figure 23 – Micro model for the Wildhammer Keeper card.

Source: By the Author.

The *micro* model is used according to the application needs. For example, the model depicted in Figure 23 represents cards with isolated symbols, thus assigning them with other cards. This kind of representation stores a vaster amount of semantic information using less storage space. By contrast, a *macro* model indicates a wide view of the game and uses less information about objects in the game. For example, the three cards showed in Figure 24 are represented by the symbols: *card 1*; *card 2*; and *card 3*. As showed in Figure 24, the card *Wildhammer Keeper* now is represented with one symbol called *card 1*, thus resulting in a simpler representation that costs more storage space.

Figure 24 – Macro model for three *Hearthstone* cards.

Source: By the Author.

9.3 Numeric model for symbols

In order to model numerically a *Hearthstone* card, it is proposed that symbols need to be mapped into variables that work inside a numeric range. For example, the *Wildhammer Keeper* card depicted in Figure 23 can be represented with the feature vector $I = \{a, b\}$, where a will be equals to 1 if the card posses the symbol *Taunt* and equals 0 otherwise, and b will be equals to 1 if the card posses the symbol *Overload* and equals 0 otherwise. This kind of codding is done for all symbols available in the game from all cards, thus generating the feature vector $F = \{f_1, ..., f_n\}$, where n is the total number of features from all cards in the game and each $f_i \in F$ is a binary variable that portray the presence or absence of a symbol on the object that are being represented. Other attributes can also be represented, like the attack and health of minions. In order to represent those

attributes the vector F can be enhanced, where $F = \{f_1, ..., f_n, z_1, ..., z_m\}$, each $z_i \in F$ is the numeric value of the symbol that represents a card attribute and m being the total number of attributes.

9.4 Attribute curves model

Attributes and symbols can be represented with the proposed numeric model for cards, however each *Hearthstone* battlefield can have more than just one card. To represent more than one card, it is proposed that attribute curves should be used with a numeric model, where an attribute curve is a histogram of the attributes of all cards present on a battlefield. The particular idea in using this model is to generalize as much as possible when representing cards inside an environment, thus the high cognition feature vector can be exploited by the ART system generalization mechanism. An attribute curve is composed by an attribute vector, of the class cls, $AT_{cls} = \{at_1, ..., at_g\}$, where each $at_i \in AT_{cls}$ represents the sum of the attribute i from a class cls for all cards on the battlefield. If creating an attribute curve AT_h to represent the *health* for each card on a player's battlefield, thus each $at_i \in AT_h$ should represent the health sum for each health level i.

9.5 Compact environment model

A compact environment model is the collection of all the visible entities in a *Hearthstone* battlefield and it is defined by the variable sets E and A, where E represents the set of environment variables and A the possible actions. The environment E is coded as a set of 8 variables, where $E = e_1, ..., e_8$. The variables $e1$ and $e2$ indicate the summarized amount of health and armor for the player and opponent, respectively, e_3 and e_4 represent the total number of cards in the hand's of each player, e_5 and e_6 the total amount of minions on each battlefield and the variables e_7 and e_8 represent the card score for all the minions in both battlefields respectively. This model of environment coding was obtained by analyzing both the battlefields and extracting the variables that are directly shown to a player, but it is important to note that not all cards and actions inside the game have full visibility, thus the game can be considered as a POMDP. The lack of information when coding environments enables further generalization by *HearthBot* when representing more than one environment, thus completing a concise categorization of a set of environments.

9.6 Full compact action model

The compact model, that uses a spectrum coding, is represented by events that happen in a time flow interacting within game components. Actions are coded in the proposed solution as a set of events that occurs on a Hearthstone environment. Each event is represented by card's semantics and extracted, in most part, from its text as illustrated in Figure 25, where the Envenom card's text was transformed into an event called *Double Attack* that targets an *Ally weapon* through interpreting the text *"Double your weapon's Attack this turn"*. Events can interact with others and consequently enable cards to influence how the game will behave, for example, if a player plays the card A it will trigger its events that will possibly interact with game components and other cards. In order to represent how events interact with other components, specially other cards, each action can be coded as a vector composed by a source and by a target. For example, if a player plays card A, then it will interact with card B, thus leading to the creation of an action vector (A, B) that represents the action that was taken. All events that can occur by playing a card A are encapsulated inside that card, this resembles a compact model to avoid the representation of all possible events that exist inside the cards with semantic structures like trees.

Figure 25 – Envenom card from *Hearthstone* with its description text "*Double your weapon's Attack this turn*".

Source: By the Author.

However, coding sources and targets, is impossible to identify what kind of events happened between a card A and a card B, thus it is useful in coding event types alongside the target and source, as a delegated event model used in many computer programs like

operational systems, thus turning the action pair into a tuple, ordered list of elements, of the form (a_i,s_i,t_i), where a_i is the event type, $s_i \in [1, 1125]$ the source of the action for all possible cards on the game and $t_i \in [1, 1125]$ the target for all possible cards on the game. It is hard to know exactly how many event types exist in the game, thus it is unfeasible to code all of them inside a_i. To tackle that, each possible value of a_i was summarized in categories of events, where four main categories were created and defined as follows: 1) play card; 2) battlecry; 3) discover; 4) physical attack. When using categories, each a_i will be inside the interval [1,4].

9.7 Partial behavioral action model

A behavioral action model is composed of low level behaviors, where each behavioral action depicts an entrance in the action vector $A = \{a_1, ..., a_l\}$, being each $a_i \in A$, a composite action and l the total number of composite actions. Each a_i in A is a binary value that represents if the actions are being used or represented by A, thus the reactive and temporal learning models from the FALCON architecture can be used. Each behavior from a behavioral model is defined by a symbolic system, algorithm, created by a specialist and that helps an agent in deciding what to do when performing.

For the proposed *Hearthstone* model, each behavior from A is defined as a greedy policy, where the best action is selected considering a local impact on the game. Furthermore, the proposal classifies each possible *Hearthstone* behavior with the categories: *physical attack*; *battlecry*; *discover*; *equip weapon*; *hero power*; *spell*; *play card action with microfeature*; *play card action without microfeature*; *face behavior*; and *end turn*. Moreover, the *play card action without microfeature* and *play card action with microfeature* are responsible in playing cards that contains or not, respectively, a determined microfeature. All the behaviors are summarized in Table 1.

9.8 Action observability and selection

In contrast to the proposed compact model, the behavioral model assumes that an algorithm selects what actions is the best for a behavior type. With this model, at each State, an agent will have to select the best behavior to perform, however each behavior can handle a list of actions from its type. For example, inside the *Physical Attack* behavior there can be various sources and various targets, thus it has a list of actions where it can

Table 1 – Proposed behavioral model for *Hearthstone* with behaviors types and its respective goals.

Behavior	Goal
Physical Attack	Select a card from the player's battlefield. Select and attack the target with the selectet card.
Battlecry	Select a card with battlecry and play it.
Discover	Select one card from the discovery mechanics.
Equip Weapon	Select a weapon card and use it.
Hero Power	Use hero power.
Spell	Select a spell card and use it.
Play card without MF	Select a card that does not contain a MF from a list of available MFs and play it.
Play card with MF	Select a card that contain a specified MF from a list of available MFs and play it.
Face	Give the maximum amount of damage to the enemy hero with all the player have.
End turn	Finishes the player turn.

Source: By the Author.

pick one to be performed by an agent. By doing that, it can not be guaranteed to select proper actions.

9.9 Extracting microfeatures for *Hearthstone*

There are two major behaviors from the proposed model that give the ability to an agent to play cards with certain microfeatures. In order to select microfeatures, each card from the game is represented with semantic networks. All networks from all cards are created from frames obtained directly from the *Hearthstone* simulator called *Metastone*. This model of coding features is used to represent cards as semantic objects to use their symbols to filter what cards are allowed to be used at each behavior. All microfeatures from all cards in the game were obtained through a descendant parser that seeks information from each card inside their frames for later being assembled in its semantic network. According to the complete information acquired from this process, the game posses 664 microfeatures that are shared between all cards. From all the 664

microfeatures, 42 were selected through feature threshold, that selects features with a variance greater than a threshold t for a microfeature mf_{count} count variable in all cards. The result of the microfeatures extraction method is a semantic network for each card available on the game. It resembles a histogram for each card microfeature list inside its semantic network.

Cards are delegated through all behaviors from Table 3 by checking if at least one of their microfeatures, that are present inside their semantic network, is compatible with the behavior type. If compatible, the behavior is said to be able to perform that action and is used by an agent to guide itself through its *Time Flow*.

9.10 Utility value of a *State* for *Hearthstone*

All the proposed *HearthBot* versions presented on the subsequent sections to perform, an utility function needs to be used to evaluate environments obtained from a State. The proposed utility function for *Hearthstone* was created based on a base value function obtained from the *Metastone* simulator version 1.2.0, where a perceived environment model is evaluated according to the value of each minion and player stats. The base utility function, obtained from *Metastone*, is formed as,

$$u = \begin{cases} 1 & \texttt{if the enemy is destroyed} \\ -1 & \texttt{if the player is destroyed} \\ \Delta hp + \Delta 3hand + \Delta 2minions + \Delta score & \texttt{otherwise} \end{cases} \tag{55}$$

where Δhp is the difference from player 1 hp to player 2 hp, $\Delta hand$ is the difference from player 1 total cards in hand from player 2 total cards in hand, $\Delta minions$ is the difference from player 1 total minions from player 2 total minions, and $\Delta score$ is the difference from the minion score calculated for all player 1 minions and all player 2 minions. It is essential to note that, the variable *score* is calculated with Equation 56 and it embodies the raw value of a minion,

$$score = \sum_i (base_i + 2t_i + 0.5w_i + 1.5d_i + sp_i + 1e_i + 1s_i + 1.5ut_i \times base_i) \tag{56}$$

where $base_i = (minionhp)_i + (minionattack)_i + (minionspelldamage)_i$, and where t_i, w_i, d_i, sp_i, e_i, s_i, and ut_i are binary values that represents the presence or absence of the symbols *taunt*, *windfury*, *divineshield*, *spelldamage*, *enraged*, *stealth*, and *untargetablebyspells*, respectively, on a evaluated minion i.

The base function represented by Equations 55 and 56 works inside an infinite range, since minion enhancements, from the game, are not countable. Furthermore, this utility function can not be used as presented by Q-learning methods, since it does not represent a magnitude in which a Q-value should be adjusted with.

To tackle that problem, it is proposed that the utility value should be computed with Equation 57,

$$\zeta_{t(i)} = \begin{cases} 1 & \texttt{if the player wins the game} \\ 0.5 + \dfrac{norm(u_{t(i)}) - norm(u_{t(i+1)})}{2} & \texttt{otherwise} \end{cases} \tag{57}$$

where if the player wins the game from the move performed on the State $t_{(i)}$ from a State i, then it will return 1, otherwise, it will return a number around a base reward equals to 0.5, where *norm* is a feature scaling function represented by Equation 58, and $u_{t_{(i)}}$ and $u_{t_{(i+1)}}$ being the computed utility values from the State $t_{(i)}$ and the newly obtained State $t_{(i+1)}$,

$$norm(r) = \frac{min + r}{max} \tag{58}$$

where r is the computed reward that will be normalized, min the minimum observed lower boundary for sampled u values and max being the maximum observed boundary for sampled u values.

9.11 Discussion

The hearthstone environment model is divided in a compact environment and attribute curves models. In this research, the compact model and attribute curves model are mixed together for HearthBots to understand a State. Moreover, the HearthBots systems were deployed in various versions varying the model used to code its actions. By deploying bots with different action coding styles, it was viable to verify the behavior of an

agent that learns from scratch, by using a spectrum action coding, or that learns with the help of small behaviors, by using a behavioral action coding. It is essential that rewards need to be calculated, thus the utility value proposal is used to perceive rewards from performing actions for all HearthBot agents. Furthermore, the Q-Learning is also adjusted by proposed Q-Learning adjustment function for Hearthstone. Peculiarly, HoningStone has been created with the HearthStone game definition itself, since it structures its information as an ontology that is able to represent events and interaction between cards through them. All the aforementioned systems are presented in Chapter 10.

10 HONINGSTONE AND HEARTHBOT SYSTEMS

This chapter presents all deployed solutions for HearthStone. It is divided in 5 sections, where section 1 presents the HoningStone system. Next, in Section 2 the HearthBot is presented. Furthermore, Section 3 introduces the T-HearthBot. In addition, in Section 4 the CTH-HearthBot, a Creative Hearthbot, is presented in details. Moreover, the CTUH-HearthBot, a Creative Hearthbot based on the UAM, is introduced in Section 5. The other variants of HearthBots were not described into this chapter since they are made by simple changing the learning algorithm or combining techniques deployed into the T-HearthBot, CTH-HearthBot or CTUH-HearthBot.

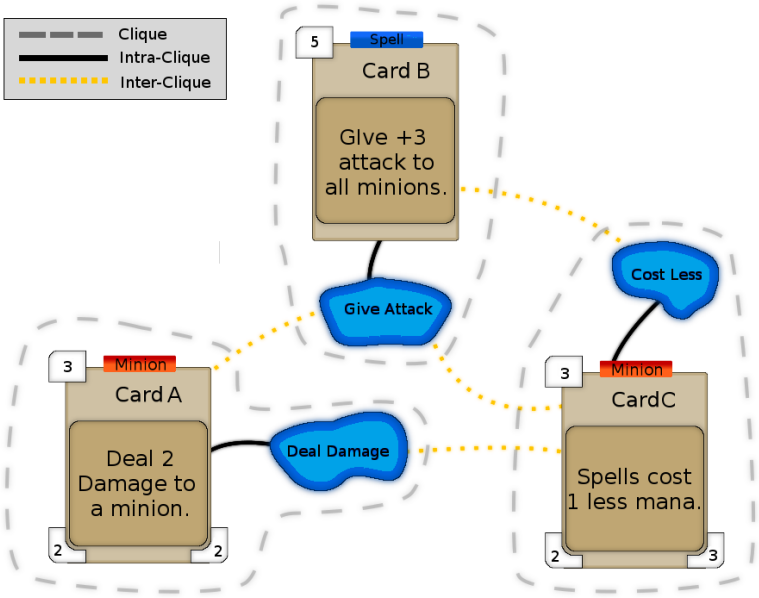
10.1 HoningStone

In this section, it is presented HoningStone, a *Hearthstone* model for the GRASP Honing system, proposed in Chapter 7, for the generation of creative card combos for *Hearthstone*. The proposed symbolic Honing Network for *Hearthstone* is composed by two forms of microfeature associations: intra and inter-clique. For the proposed example depicted in Figure 26, bridges are seen as inter-clique relationships and are represented by the dash yellow lines. In contrast, cliques from a super node are represented by intra-clique relations. Each card from the proposed network represents a super node composed by microfeatures, where each microfeature are cards semantics that were extracted from the card behavior described in its text. The example showed in Figure 26, represents three cards, A, B and C, and each indicate a super node from the Honing Network.

The main problem of the Honing GRASP model is that it need to be conceived by an specialist, consequently each microfeature extracted from cards text was given by an specialist.

In Figure 26, the card A is represented by the text *Deal 2 Damage to a minion*, where the node microfeature *Deal Damage* was created to constitute its semantics. The created microfeature is related to card C, since the target of *Deal Damage* is a minion. The proposed Honing network ignores values and codes semantics in order to simplify the random construction generation process and evaluation. As a result from the interaction between elements in the proposed honing network for *Hearthstone*, the GRASP honing will be able to identify which cards combines with others by recognizing their semantic relationship.

Figure 26 – Hearthstone HoningNetwork model.



Source: By the Author.

10.1.1 Creativity Metric for GRASP evaluation

In this research, a simplified version of a creativity metric based on the Bayesian surprise and an efficiency metric, proposed by (JUNIOR et al., 2016) is used to guide the HoningStone system. A model to represent each combo was also defined by (JUNIOR et al., 2016) and it calculates both metrics. In this model, called Regent Dependent, each combo is composed of cards, which in turn has effects. Each effect is modeled as a pair $P(ability, target)$ which has a value v. For instance, "destroy 2 minions", is represented as $P(destroy, minion) = 2$. Hearthstone produces 190 distinct pairs when combining all abilities and targets from the existing card set. These pairs exhibit all kinds of effects that a combo can deliver. Thus, each combo can be modeled as an array of these 190 pairs, where the value v of each pair, which is extracted from the effect, indicates the intensity of the respective effect. If an effect appears more than once in a combo, its values are added and stored on its respective pair. Based on this combo representation, the surprise and efficiency metrics are then described.

Firstly, the Bayesian surprise enables to evaluate how much new is an artifact compared to known ones. In particular, the surprise of a combo pair c_i is calculated by equation 13, where μ_i is the new observed value of that pair, σ^2 is the variance and $\bar{m}$ is the average, both calculated based on all known combos pair c_i values. The complete

surprise of a combo is the sum of all c_i, normalized using the feature scaling technique to yield its value to the range [0,1].

The efficiency metric proposed by (JUNIOR et al., 2016) is based on a synergy graph, created by an specialist, for each combo. However, in order to assemble a similar structure without handcrafted structures, in HoningStone the efficiency of a combo is measured through k-Nearest Neighbors algorithm using a database of combos and their respective win rate. Each combo from this dataset posses an associated win rate calculated by the number of *wins* divided by the number of *matches* extracted from community web sites. A new combo win rate is estimated by averaging the win rate of its neighbors returned by the k-Nearest Neighbors, which uses the euclidean distance. The efficiency is represented between [0,1], thus not requiring normalization.

Lastly, the creativity metric is calculated by equation 59, where efficiency e and surprise metric s are added representing. The second expression is used to avoid that artifacts with high efficiency and low surprise, or vice-versa, score a high creativity, by penalizing an artifact creativity value as the difference of efficiency and surprise increases.

$$creativity(e, s) = e + s - \sqrt{|e - s|} \tag{59}$$

This creativity metric, with range [0,2], guides the analytic mode to add cards which make a combo more surprising and at the same time more efficient.

10.2 HearthBot

In this Section is presented *HearthBot*, a *Hearthstone* agent that can autonomously learn and play *Hearthstone*. The proposed *HearthBot* is an *Hearthstone* agent created to verify the win rate convergence of the proximity ANN, proposed in Chapter 8, where its mechanisms are used to predict actions according to an observed environment.

10.2.1 Adaptive Neural Network architecture for HearthBot

The architecture of *HearthBot* is depicted in Figure 27, where the environment E is composed by the proposed *Compact environment model*. The compact model was used since a graphics processing unit bus transfer speed can not handle large amounts of data transfers fast enough, where it can cause a delay when processing States from a *Time Flow*. Furthermore, the proposed model follows the FALCON topology, but it does not uses any temporal learning technique since it is used to evaluate the proximity ANN.

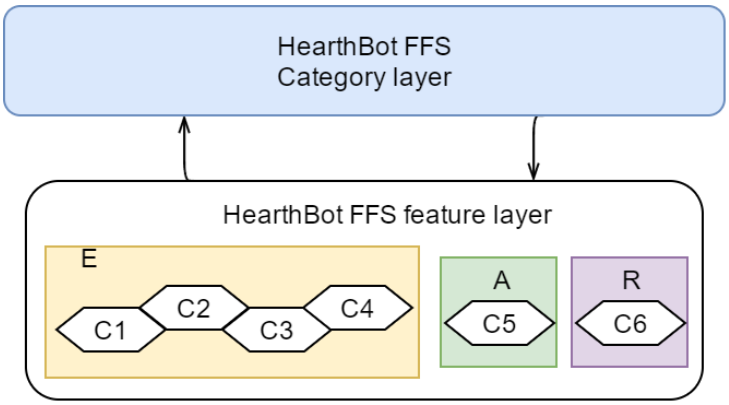
In order to assemble *HearthBot*, all fields from the environment model were decomposed into four sub fields c^1 to c^4. By decomposing the environment into more than one field it's expected to prevent variables to being underrated when categorizing the received stimulus. The underrating happens when variables working within different ranges, inside the normalized interval [0,1], are treated by the categorization mechanism as working on the same range, thus resulting in inefficient categorizations.

To represent actions, a *full compact action model* is used, since the behavior of the agent will not be influenced by the lack of observability from a *partial behavioral model*. The action field is represented by c^5, but it alone does not resemble a FALCON architecture. To tackle that, a reward field c^6 is used to model a virtual POMDP as a FALCON input model.

10.2.2 HearthBot as a Metastone interface

HearthBot architecture is illustrated in Figure 28, where it acts as a *Metastone* interface, that communicates with the proposed ANN, inside the GPU, through the *Send* and *Receive* interfaces. The *Send* interface is used to request the processing of the environment E, Action A and Reward R coded by the fields c^1 to c^6, and the *Receive* gives a response to *HearthBot* as a prediction. This received prediction is then used by

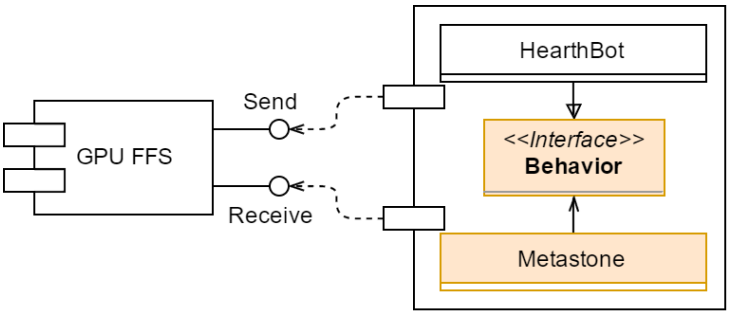
Figure 27 – *HearthBot* architecture for the proximity ANN.



Source: By the Author.

HearthBot to make a decision of what action to perform next for all turns in a game until someone gets defeated.

Figure 28 – Component diagram of *HearthBot* as a *Metastone Behavior* interface.



Source: By the Author.

10.2.3 HearthBot algorithm

HearthBot agent 3-step routines are performed through the *Behavior* interface, as depicted in Figure 28. Its algorithm, showed in Figure 4, relies in sensing the *Hearthstone environment* and coding it through an environment, action and reward coding mechanisms introduced in Chapter 9. The input parameter e represents the perceived environment from a *Hearthstone* game, a representing all the possible actions that the agent can execute at a turn, *reasoner* as the proposed multi-channel ANN, lr as a local reasoner to simulate the executed action, and *learning* as a boolean to identify if the network is in learning mode.

In order to achieve a prediction, all the variables are coded, as a message string

to be sent to the ANN input variables at I in order to be normalized and copied to the activity vector X. After sending the message, *HearthBot* can obtain a response string using the *Receive* interface from the ANN on the GPU if the previous sending message was in prediction mode.

Algorithm 4: *HearthBot* as a *Metastone Behavior* interface

Input: Environment E, PossibleActions A, ANN *reasoner*, LocalReasoner lr, Boolean *learning*
Output: Action to use *act*

```
1: Action act = NULL
2: if (learning) then
3:    act = SelectAction(A)
4:    Reward r^old = GetReward(E)
5:    Reward r^new = lr.SimulateAction(act)
6:    Reward r^real = r^new − r^old
7:    reasoner.Send(E, act, r^real, learning)
8: else
9:    Rewards R = ∅
10:   for a_i ∈ A do
11:      reasoner.Send(E, a_i, r^real, learning)
12:      Reward r = reasoner.Receive() // call local reasoner proposed in Section 9.10
13:      R.add(r)
14:   end for
15:   act = max(A, R)
16: end if
17: Return act
```

Source: By the Author.

When in learning mode, the algorithm selects an action a_i from an action selection policy through the function *SelectAction*, this action is subsequently used to calculate the initial reward r^{old}. On the original FALCON, the immediate reward for an executed action a_i is calculated by a direct response from an environment, but this could lead to false rewards. It stems from the fact that an immediate reward does not consider time changes between the environment E_t to E_{t+1}. This research overcomes that by calculating the delta reward from the current E_t to the next E_{t+1}, thus leading to more precise immediate reward calculations for an executed action a_i. The full reward for the executed action a_i is calculated through the help of a local reasoner. This local reasoner acts as a local solution searcher and it is a function from *Metastone* that permits to simulate actions without changing an environment E. After the new reward calculation in r^{new}, the real one can be

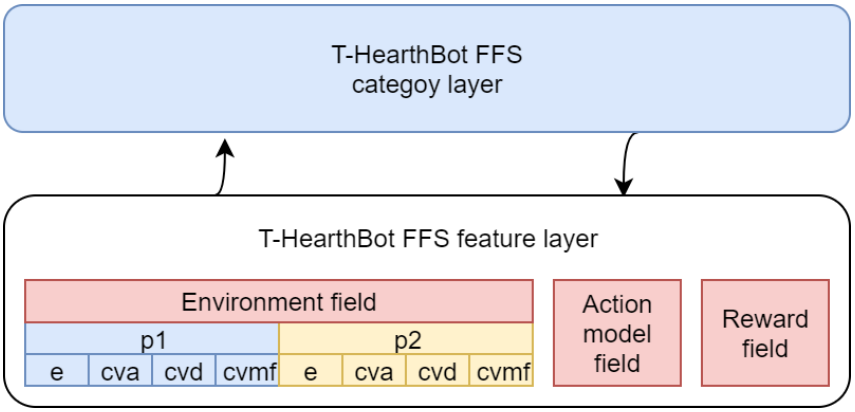
calculated by $r^{real} = r^{new} - r^{old}$. When $r^{real} > 0$, then there was a positive change in favor to the agent, otherwise the change was negative.

If not in learning mode, the *HearthBot* algorithm will try to predict the best reward for all the possible actions A. This mode of the algorithm resembles the FALCON, one proposed by (TAN, 2004) to control agents in real time. The basic procedure starts predicting all the rewards for all the possible actions $a_i \in A$ on the environment E, then selecting an action with the maximum predicted reward $r_i \in R$ through the function $max(a_i, r_i)$ for all $a_i \in A$. This action is then, returned to the agent to be executed as a decision. It is critical to note that all the rewards are calculated by a heuristic function provided by *Metastone*. This heuristic function considers the variables codified by E to calculate the score based on the proposed utility value by Equation 57.

10.3 Temporal HearthBot

In this Section is presented the T-HearthBot, where it encompasses the proposed environment and action models used with the Q-Learning or a Reactive model for a FALCON architecture. Since the *T-HearthBot* uses a temporal technique it is expected a low cognitive code growth, thus it is not mandatory for *T-HearthBot* to be deployed directly into a graphics processing unit as the *HearthBot* in order to obtain a hardware acceleration when performing. As depicted in Figure 29, the *T-HearthBot* is composed by three main fields, as a FALCON architecture, where the environment field is composed by four models for each player. The four models used for *T-HearthBot* are: e, obtained from a *Compact environment*; cva, obtained from *Attribute curves* for all minions attack; cvd, obtained from *Attribute curves* for all minions defense; and $cvmf$, obtained from a *Numeric model for symbols* extracted from semantic networks for all minions.

Figure 29 – Field architecture for *T-HearthBot*.



Source: By the Author.

The proposed *T-HearthBot* is deployed with the two action models: *Full compact* and *Partial behavioral*. When using each of the proposed models, *T-HearthBot* performs as a FALCON system, thus a reward field is also assembled in its feature layer. Each field from the *T-HearthBot* is coded with ART neurons, but if the used action model is the *compact* one, then the proximity neurons, from the proximity ANN proposed in Chapter 8, are used instead to prevent over generalization from an action spectrum model.

10.3.1 Temporal Reactive Algorithm for T-HearthBot

The reactive temporal algorithm for T-HearthBot, presented in Algorithm 5, is assembled as a reactive FALCON, obtained from (TAN, 2004), and it is divided into two steps: *performing* and *learning*. Its input parameters are an environment E, an available action set A, the operating ANN, a local reasoner lr and an exploration threshold e. Its output parameters are an environment observation $currentEnvVec$, the selected action from the process inside the variable *selectedAction*, the calculated rewards r_{new} and r_{old} and the operating ANN.

Algorithm 5: Reactive FALCON algorithm for *T-HearthBot*

Input: E, A, ANN, lr, e
Output: $currentEnvVec$, *selectedAction*, r_{new}, r_{old}, ANN

```
1: r_old = r_new
2: explore = rand() // rand function generates a random number between 0 and 1
3: currentEnvVec = buildEnvironmentVec(E)
4: selectedAction = null
5: if (explore < e) then
6:    selectedAction = randomAct(A) // randomAct function selects a random action
7: else
8:    while (selectedAction == null) do
9:       actionToPredictVec = directAccess()
10:      rewardVec = {1.0, 0.0}
11:      ANN.setActivity(currentEnvVec, actionToPredictVec, rewardVec)
12:      prediction = ANN.predict()
13:      selectedAction = maxAct(prediction.actionField)
14:      if (invalid(selectedAction)) then
15:         // verify if the selected action can be performed
16:         selectedAction = null
17:      end if
18:      if (A != contains(selectedAction)) then
19:         actionToPredictVec = generateAvailable(A)
20:         ANN.resetLast(actionToPredictVec)
21:         selectedAction = null
22:      end if
23:   end while
24: end if
25: // call local reasoner proposed in Section 9.10
26: r_new = lr.simulateAction(selectedAction)
```

Source: Adapted from (TAN, 2004).

The performing step is responsible for retrieving a predicted action mask from a

given observed environment State in E. This action mask will be predicted if the *explore* variable goes bellow the exploration threshold e. In order to predict an action mask, the $rewardVec$ is configured as [1,0] to get the max reward prediction. Furthermore, the $actionToPredictVec$ represents what action will be predicted and it is assembled with the direct access method through function *directAccess*. The *setActivity* function is used to assemble the activity in order to fulfill a prediction for the reward field. Conclusively, a prediction is achieved when calling the *predict* function from the ANN.

The while mechanism in Algorithm 5 is used to prevent non existent actions into being selected from the prediction mechanism, thus if that happens the $resetLast$ function will force the network into replacing the most recent predicted neuron action mask with a valid one, created with the *generateAvailable* function. It is essential to note that the variables r_{old} and r_{new} are configured at the start and end of the algorithm, respectively, where the r_{new} variable is obtained through the utility function, described in Section 9.10, and is calculated within a simulator.

The *learning* step is performed after the *performing* step, since it is bounded to the *performing* step output parameters. The *learning* step is responsible to call the routines that will permit to perform the neuron *reinforcement*, *erosion*, *decay* and *prunning* described at Section 6. In order to perform any neuron *reinforcement* routines, the agent needs to suffer a positive feedback change. This feedback change is controlled by the condition $r_{new} > r_{old}$, where if true it will write, into the ANN, the received reward, r_{new}, and the performed action, *selectedAction*. If the feedback change condition returns false, then the agent will reset the action mask from the observed environment stored in $currentEnvVec$. The reset will ensure that the *selectedAction* is a bad choice when interacting within E.

The *reinforcement* routine is called by the *reinforcement* function inside the ANN and the *erosion* is called by the *erosion* function, where either need to be preceded by the constructions of the activity vectors inside the ANN $F1$ feature layer through the *setActivity* function. At the end of the algorithm, the neuron *decay* and *prunning* are called by the *neuronDecay* and *neuronPrunning* functions inside the ANN.

10.3.2 Temporal Q-Learning Algorithm for T-HearthBot

The Q-learning algorithm for the *T-HearthBot* incorporates the temporal learning method and was assembled based on the pseudo algorithm presented in (WANG; TAN,

Algorithm 6: Reactive FALCON learning algorithm for *T-HearthBot*

Input: $currentEnvVec$, $selectedAction$, r_{new}, r_{old}, ANN
Output:

```
1: if (r_new > r_old) then
2:    // positive change block
3:    actionToPredictVec = buildAct(selectedAction)
4:    rewardVec = {r_new, 1 − r_new}
5:    ANN.setActivity(currentEnvVec, actionToPredictVec, rewardVec)
6:    ANN.reinforcement()
7: else
8:    // negative change block
9:    actionToPredictVec = buildResetAction(selectedAction)
10:   rewardVec = {1 − r_new, r_new}
11:   ANN.setActivity(currentEnvVec, actionToPredictVec, rewardVec)
12:   ANN.erosion()
13: end if
14: ANN.neronDecay()
15: ANN.neuronPrunning()
```

Source: Adapted from (TAN, 2004).

2015). In this research it was divided into three parts: *sensory to action*; *action inhibition*; and *learning*. As the reactive model, it receives an environment E, a list of possible actions to perform in A, the operating ANN, the local reasoner lr, and the exploration threshold e. It outputs a calculated immediate reward r, the predicted Q-value $Qvalue$, a copy of the environment E inside s_{out}, and a copy of the selected action inside a_{out}, where all output parameters are used on the subsequent steps of the algorithm.

The algorithm starts with the *sensory to action* step, described in Algorithm 7, where it will decide in exploring or exploiting. If exploiting, it will take a random action based on the threshold e, otherwise it will perform an action inhibition in order to select the best action for the observed environment E. The functions *buildEnvironmentVec* and *buildAct* are responsible to assemble the semantic symbols into feature vectors as illustrated in Figure 12.3.1. Furthermore, the *calculateQValue* function calculates the Q-value of the obtained prediction for a given activity vector configured by *setActivity* function. At the end of the algorithm, the selected action, a_{out}, predicted Q value, $Qvalue$, immediate reward calculated through *simulateAction* function, r, and used State, s_{out}, are configured as outputs for the next step.

The *action inhibition* step is responsible for selecting the best observed Q value for a given observation E and possible actions A. It receives all output parameters obtained

Algorithm 7: FALCON Q-learning sensory to action step

Input: E, A, ANN, lr, e
Output: r, $Qvalue$, s_{out}, a_{out}

```
explore = rand() // rand function generates a random number between 0 and 1
if (explore < e) then
    selectedAction = randomAct(A) // randomAct function selects a random action from A
    currentEnvVec = buildEnvironmentVec(E)
    actionToPredictVec = buildAct(selectedAction)
    rewardVec = {1.0, 1.0}
    ANN.setActivity(currentEnvVec, actionToPredictVec, rewardVec)
    prediction = ANN.prediction()
    maxQ = calculateQValue(prediction.rewardVec) // calculate the Q value from the prediction
else
    ActionInhibition(IN E, IN A, IN ANN, OUT selectedAction, OUT maxQ)
end if
a_out = selectedAction
Qvalue = maxQ
r = lr.simulateAction(selectedAction) // call local reasoner proposed in Section 9.10
s_out = E
```

Source: Adapted from (WANG; TAN, 2015).

from the *sensory to action* and it outputs a predicted Q-value inside $Qvalue$ and the selected action inside $selectedAction$.

The *inhibition* step starts by configuring the observed environment into $currentEnvVec$ variable through *buildEnvironmentVec* function. After its initialization, it will seek for each action $a_i \in A$, the best predicted Q value and stores it into $maxQ$. The selected action is stored inside $selectedAction$ variable and it will be used as an output for this step. The *buildAct* function is responsible for assembling the action vector, as in step 1, and the *prediction* function will force the working ANN to predict the Q value according to the FALCON architecture. At the end, the algorithm configures the max predicted Q value into $Qvalue$ and uses it as an output parameter alongside with the $selectedAction$, used to perform.

At the final step, *learning*, *T-HearthBot* will learn based on the temporal difference. This step is responsible for calculating the $TDErr$, temporal difference error, in order to adjust the simulated POMDP weights from each performed action during its performance. It receives the output of the *sensory to action* as input parameters and also it has three control flags: $useReward$, responsible to decide if it calculates the immediate reward or not;

Algorithm 8: FALCON Q-learning Action Inhibition

Input: E, A, ANN, $Qvalue$, selectedAction
Output: $Qvalue$, selectedAction

currentEnvVec = buildEnvironmentVec(E) // assemble environment vec
$maxQ = -\inf$
selectedAction = null
for (Action $a_i \in A$) **do**
  actionToPredictVec = buildAct(a_i)
  rewardVec = {1.0, 1.0}
  ANN.setActivity(currentEnvVec, actionToPredictVec, rewardVec)
  prediction = ANN.prediction()
  predictedQ = calculateQValue(prediction.rewardVec)
  if (predictedQ > maxQ) **then**
    maxQ = predictedQ
    if (selectedAction != NULL) **then**
      selectedAction = a_i
    end if
  end if
end for
$Qvalue = maxQ$

Source: Adapted from (WANG; TAN, 2015).

boundedQ, that tells if the algorithm will use the bounded Q-learning; and $ThresholdQ$, that tells if it will use the threshold Q-learning.

The *learning* step starts by calculating the error for a predicted Q value from $t(i)$ and a new Q value from $t(i+1)$, where $t(i)$ is the State at the State i and $t(i+1)$ a State from the State $i+1$ from the game's *Time Flow*. It then rescales a calculated learning value, $Qlearn$, using the bound rule or the threshold Q-learning method. At the end of this stage, the algorithm assembles the activity vector through *buildEnvironmentVec* and *buildAct* functions, and set the ANN activity through *setActivity* function in order to learn by calling the ANN *learn* function.

10.3.3 Discussion

When performing through the Q-learning algorithm, some precautions need to be taken in order to guarantee its integrity. Firstly, it is essential to develop an external mechanism that enable to store previously predicted Q values in order to calculate the steps from the *learning* stage properly. Secondly, it is important in assembling a list of possible actions to perform, since it will facilitate the algorithm mechanism in search for

Algorithm 9: FALCON Q-learning learning

Input: r, Q_{old}, Q_{new}, ANN, s_{out}, a_{out}, $useReward$, $boundedQ$, $ThresholdQ$
Output:

$TDerr = r + \gamma * Q_{new} - Q_{old}$
$\Delta Q = \alpha * TDerr$
$Qlearn = 0$
if $(boundedQ)$ **then**
  $Qlearn = Qvalue + \Delta Q * (1 - Qvalue)$
else
  if $(ThresholdQ)$ **then**
    $Qlearn = Qvalue + \Delta Q$
    $Qlearn = min(1, Qlearn)$
    $Qlearn = max(0, Qlearn)$
  end if
end if
envToLearnVec = buildEnvironmentVec(s_{out})
actionToLearnVec = buildAct(a_{out})
rewardToLearnVec = $\{Qlearn, 1 - QLearn\}$
ANN.setActivity(envToLearnVec, actionToLearnVec, rewardToLearnVec)
ANN.learn()

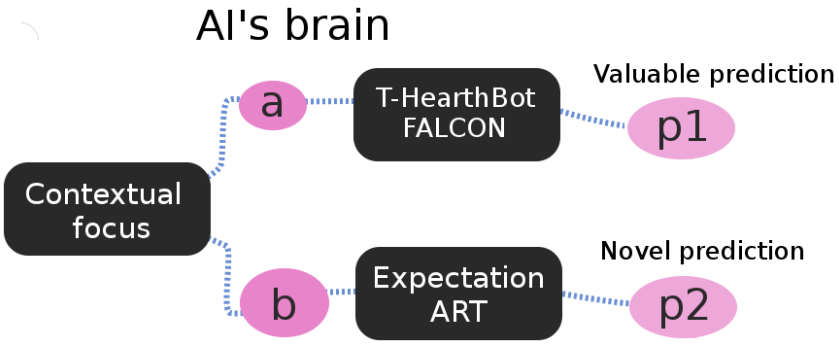
Source: Adapted from (WANG; TAN, 2015).

an action to perform inside an action vector A. And finally, it is important in establishing a main controller that controls variables like, e, driving the dynamics of all algorithms steps.

10.4 Creative Temporal HearthBot

In this Section is proposed the *CTH-HearthBot* as a combination of the proposed HARP system, from Chapter 7.2, and the proposed *T-HearthBot* from Section 10.3. The proposed *CTH-HearthBot* is based on the proposed HARP process, as depicted in Figure 30, where the POMDP FALCON is replaced by a *T-HearthBot* FALCON. The *CTH-HearthBot* assumes both proposed action coding models from Chapter 9, thus when using a *compact* model the proximity metric should be used as well.

Figure 30 – Field architecture for *CTH-HearthBot*.



Source: By the Author.

In the example showed in Figure 30, *CTH-HearthBot* received two stimulus: a and b. The a stimulus represents what environment the bot is seeing, by contrast the b stimulus is a set composed by all possible actions that the bot can perform given a State from *Hearthstone*. The prediction $p1$, obtained from the process is representing a valuable prediction, that will optimize its overall reward over time, and the $p2$, represents a novel prediction, that will lead to novel paths on the simulated POMDP.

The environment field in the feature layer on the expectation ART, from the process depicted in Figure 30, is composed by the same environment field used on the *T-HearthBot*, thus coding environments by relating them, exclusively, to one expectation neuron as proposed in Section 7.2.

10.4.1 Algorithm for CTH-HearthBot

The main algorithm for *CTH-HearthBot*, presented in Algorithm 10, is composed by one of the temporal algorithms presented for *T-HearthBot* and the structural control algorithm proposed for the HARP process. Differently from the *T-HearthBot* algorithm, it

receives an extra ANN called *E-ANN* as an input parameter. This *E-ANN* is an expectation ART and it is used by the bot to behave according to the proposed HARP.

Algorithm 10: FALCON Q-learning sensory to action step for the HARP process

Input: E, A, ANN, E-ANN, lr, e
Output: r, $Qvalue$, s_{out}, a_{out}

```
explore = rand() // rand function generate a random number between 0 and 1
if (explore < e) then
    E-ANN.setActivity(currentEnvVec)
    prediction = E-ANN.prediction()
    selectedAction = prediction.selectedAction
    maxQ = calculateQValue(ANN, currentEnvVec, selectedAction)
else
    ActionInhibition(IN E, IN A, IN ANN, OUT selectedAction, OUT maxQ)
end if
a_out = selectedAction
Qvalue = maxQ
r = lr.simulateAction(selectedAction) // call local reasoner proposed in Section 9.10
s_out = E
```

Source: By the Author.

The Algorithm 10 works in the same way as Algorithm 7, but the main difference is the structure of the explore section controlled by the condition $explore < e$, where the selected action is the most surprising one returned by the Expectation ART through the *prediction* function. The Q-value of the surprising action is calculated with the $calculateQValue$ function, since the E-ANN does not hold a virtual Q-learning table. The rest of the process works as a *T-HearthBot* algorithm, since the presented Algorithm 10 represents the starting point of reasoning. If assembling a reactive version of the *CTH-HearthBot*, then the explore section of Algorithm 5 should be replaced by the explore section of Algorithm 10, but ignoring the $calculateQValue$ function.

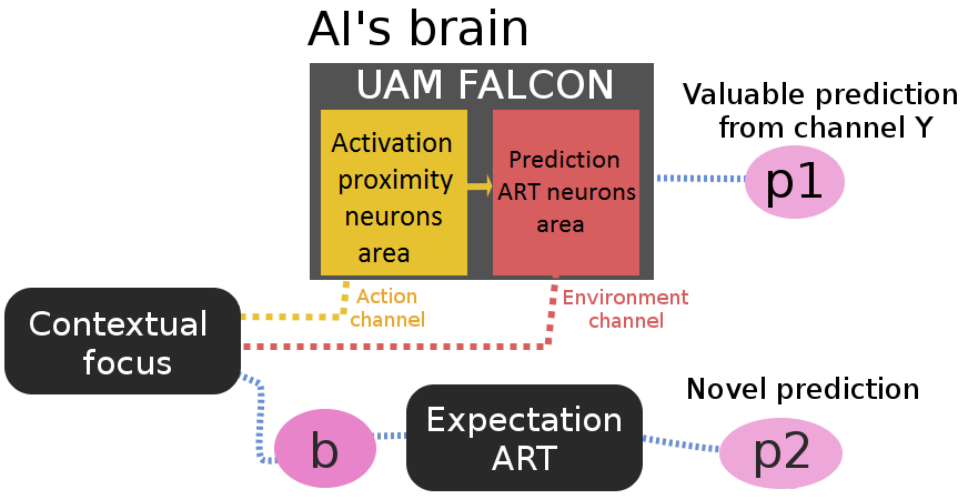
10.5 Creative Temporal UAM HearthBot

In this Section, the *CTUH-HearthBot* architecture is presented, where it is mainly composed by the proposed UAM-ANN used as the POMDP(Partially Markov Decision Process) FALCON from the proposed HARP process. This bot takes creative decisions over time considering the hierarchical model of an UAM-ANN, thus it represents the HARP process as showed in the subsequent section.

10.5.1 Architecture

The proposed *CTUH-HearthBot*s uses a UAM-ANN FALCON and a Expectation ART. As illustrated in Figure 31, the proposed bot uses one UAM-ANN that is assembled with 2 channels, where the channel 1, represented by the yellow rectangle, is responsible to hold one action field and the channel Y, represented by the red rectangle, is holding the environment and action fields. By assembling the UAM-ANN in that manner it is expected for it to behave like a FALCON architecture, where the environment and reward are predicted from channel Y.

Figure 31 – Field architecture for *CTUH-HearthBot*.



Source: By the Author.

For the this architecture, the action channel activity field is coded as the proposed environment field for the *T-HearthBot* depicted in Figure 29. The environment channel is assembled with 2 fields, where the first field is the action model field and the second one is the reward field, thus representing the FALCON architecture.

It is assumed that the action model used is the *full compact*, thus the action channel is assembled as proximity neurons, those which are activated by the proximity metric. By contrast, if using the *partial behavioral* model, ART neurons, activated by fuzzy ART metrics, are used instead. Furthermore, this kind of channel architecture assumes that there are more environments than actions, thus the search space for environment will be filtered by each represented action in action channel, thus reducing the search space and coding more environment neurons if necessary.

The algorithm for the *CTUH-HearthBot* is composed by the presented algorithm for the *CTH-HearthBot*, where the main difference is that the internal structure of the POMDP FALCON is replaced by an UAM-ANN FALCON process proposed in Chapter 8.

Part IV

Experimental evaluation, results and conclusions

11 RESULTS

This chapter presents the evaluation for the research proposals in four sections. Section 11.1 presents the HoningStone evaluations, where the proposed symbolic model introduced in Chapter 7 is evaluated. Next, in Section 11.2 the Metastone simulator, used to perform the analysis for all HearthBot versions, is presented. Next, Sections 11.3 and 11.4 present the HearthBot evaluation. All evaluations discussed in this chapter were accomplished through the deployment of the Hearthstone models, introduced in Chapter 9, and are described in details on each section.

11.1 HoningStone evaluation

The HoningStone system itself was fully implemented in C++. In order to build the honing network, a JSON file version 4.0.2 was utilized from *Hearthstonejson.com* which contains all updated cards information. We created a script that extracts all pairs and values by normalizing the effects words and handling all exceptions. It is important to note that texts present very regular patterns in Hearthstone, so it is not required a sophisticated natural language processing algorithm to extract these pairs. Once the pairs are extracted, generating the honing network is quite straightforward. Each card name (parent microfeature) is connected to its effects abilities (child microfeatures) forming cliques. Each child microfeature is then connected to parent microfeatures that have a type that matches the effect's target. This procedure is repeated for all cards.

11.1.1 Experimental design

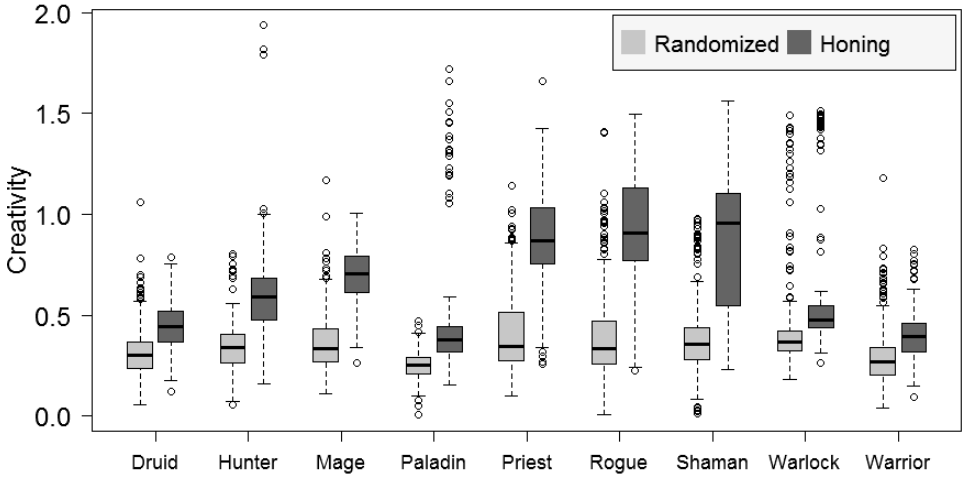
To evaluate HoningStone, 400 experiments were performed for each hero of the game, described in Chapter 9, 10 executions for each of the 40 seeds (which is approximately the number of cards specific to each hero). This generated 3600 sample points for HoningStone and an additional 3600 points for the randomized greedy approach. We set the mana cost limit to 10, which is the maximum allowed in a turn in Hearthstone, limited the combo size to a maximum of 10 cards. The window size was set to 80% based on a sensitivity analysis for both algorithms. The surprise and efficiency metrics based on a subset of 31000 distinct combos extracted from from 10000 decks played in more than 3 million matches obtained from the various public websites. Each combo inherited the win rate of the deck

it was extracted from (if the combo was presented in different decks, it was assigned with the arithmetic average of the decks win rate). In order to evaluate the efficiency of the generated combos, the K-Nearest Neighbor model returned the five closest combos win rate, which were then averaged.

11.1.2 Creative process assessment analysis

In this section, HoningStone is evaluated compared to the randomized baseline, that generates combos picking random cards, for generating creative combos. Chart 1 shows the creativity metric which combines surprise and efficiency and it ranges from 0 to 2. In the randomized version the median for all heroes is around 0.3 with a small difference between the first and third quartiles. This result indicates that the randomized version is not capable of exploring the creativity space, commonly leading to poor creative artifacts, although it generates a few outliers with moderate creativity. On the other hand, HoningStone generates combos more creative than the randomized algorithm for all heroes, where medians range from 0.4 to 1. Even when considering just outliers, the randomized version is unable to generate few combos with more than 1.5 creativity, which is done by HoningStone in most heroes.

Chart 1: HoningStone for combo generation.

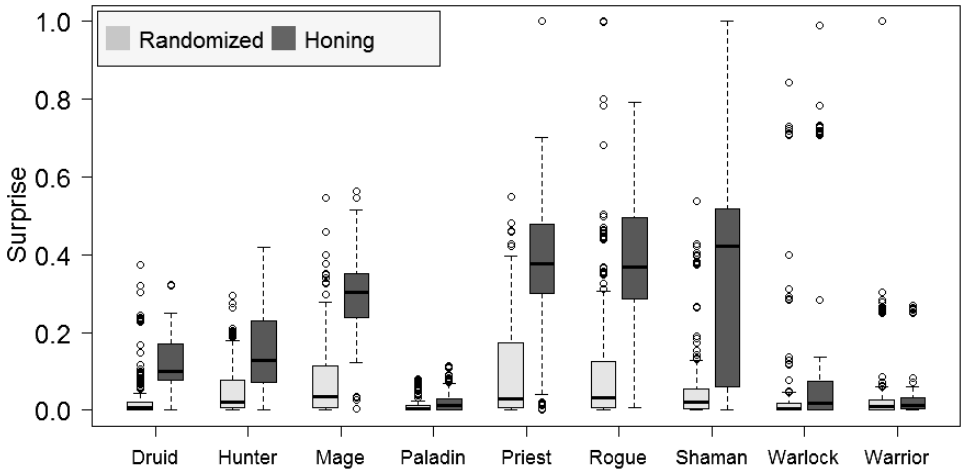


Source: By the Author.

In particular, for Paladin, which presented very low surprise, as showed in Chart 2, the randomized version could not generate combos with surprise above 0.1. After careful investigation on Paladin, is possible to observe that players tend to build the same combos based on a few effects, which are usually efficient. This can guide our system to

efficient known solutions, rather than different ones is most cases, leading to low surprise. However, HoningStone found ways to generate a few outliers with surprise near to one and consequently with creativity above one. This result suggests that further improvements in the honing network could lead to better exploration of these points.

Chart 2: HoningStone measured surprise for combo generation.



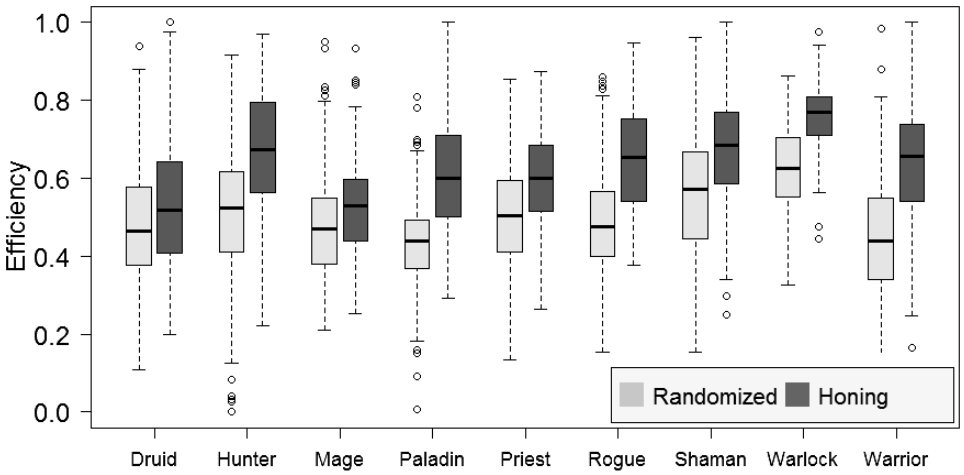
Source: By the Author.

Chart 2 shows that the randomized algorithm consistently generates combos that has near zero surprise. This result reveals that when a high efficiency combo is randomly generated, it is likely to have low surprise. Since the randomized algorithm evaluates combos based on creativity, then combos with large discrepancy between surprise and efficiency are poorly evaluated and not pursued. Due to the large combo space, the honing associative mode was key for guiding the creative process into reducing this space to cards that could actually be part of an efficient combo.

The proposed efficiency metric is based on the win rate of decks in actual games, which are ruled by a matchmaking system. Inefficient decks are not used for more than a few matches, so they are not included in our system. Due to this fact, our database of combos is based on decks that present an average 0.5 win rate, which is confirmed as depicted in Chart 3. Moreover, the K-Nearest Neighbor averages the five nearest neighbors (combos) win rate, which is good for estimating never seen combos, but can also lead to an unsatisfactory win rate for common combos presented in very different decks. Despite the efficiency model limitations, Chart 3 shows that HoningStone is able to surpass the performance of the randomized version for all heroes. In particular, for Warrior the difference between medians is around 0.2, which was fundamental to make HoningStone

combos more creative than the ones from the randomized version, as illustrated in Chart 2.

Chart 3: HoningStone measured efficiency for combo generation.



Source: By the Author.

The results showed that HoningStone can generate combos that are more creative than a greedy randomized algorithm. Although the combos generated by HoningStone are certainly new and playable from an expert point of view, their efficiency has yet to be proven in real games. Finally, the behavior generated from HoningStone indicates that the modified GRASP can still be able to guide the objective function towards a better solution, being local or global optima, after its adaptation to implement the described processes by The Honing Theory.

11.2 Metastone behavior analysis

All experiments conducted for the presented results in Sections 11.3 and 11.4, were conducted on Metastone, a Hearthstone simulator. In Section 11.3 is presented the behavior analysis of the HearthBot, deployed as a pure proximity ANN that simulates memory, and Section 11.4 presents all proposed temporal based HearthBots, that uses a behavioral and spectrum models for all proposed ANN, and the creative HearthBots, deployed with the HARP. All HearthBots were evaluated in terms of win rate, total victories divided by the total number of matches, as explained in Sections 11.3.1 and 11.4.1. For the temporal HearthBots, it was analyzed also in terms of general behavior attribute curve, where the average of each collected attribute was measured, as explained in Section 11.4.1. The selected attributes for evaluating the general performance in Metastone were: Minions played; fatigue damage; damage dealt; spells cast; weapons played; mana spent; weapons equipped; cards played; armor gained; cards drawn; hero power used; turns taken; and healing done. All extracted attributes were directly obtained from Metastone functions and are used to evaluate the general behavior of the agents, thus they do not constitute a quality measurement but rather a way to investigate in what attribute the agent is spending most of its efforts.

In order to evaluate all HearthBots competing in the game, the decks described by Table 2 were selected to play with. These decks are legendary ones selected from recent top ranked players and they are as diverse as possible regarding their strategies in order to explore the proposals capabilities in playing the game. These decks were grouped into three types of strategy: Defensive, that focus in defending the hero and battlefield; Aggressive, that focus in destroying the opponents minions and hero; and Hybrid, a mixed strategy between defensive and aggressive.

Moreover, more nine decks were selected in order to serve as unobserved decks, composed by data not used during training, that HearthBots play against in order to evaluate their capability in generalizing for environments never seen before. Those decks are also competitive ones, described in Table 3, and represent all nine heroes of the game varying strategies between hybrid, aggressive and defensive.

Metastone posses two major agents; one is based on the MCTS; and the second one is based on a board control greedy strategy and it is called Board Control Greedy (BC-Greedy). The MCTS based Metastone agent plays by selecting actions with a randomized

Table 2 – Deck selection for *HearthBots* evaluation.

Hero	Deck name	Strategy
Warrior	Control Warrior	Defensive
Warlock	Zoolock	Aggressive
Druid	Midrange Druid	Hybrid
Priest	Control Priest	Defensive
Rogue	Malygos Rogue	Aggressive
Mage	Tempo Mage	Hybrid
Shaman	Aggro Shaman	Aggressive
Hunter	Face Hunter	Aggressive
Paladin	Secret Paladin	Defensive

Source: By the Author.

Table 3 – Deck choices and the respective strategies that serves as unobserved data

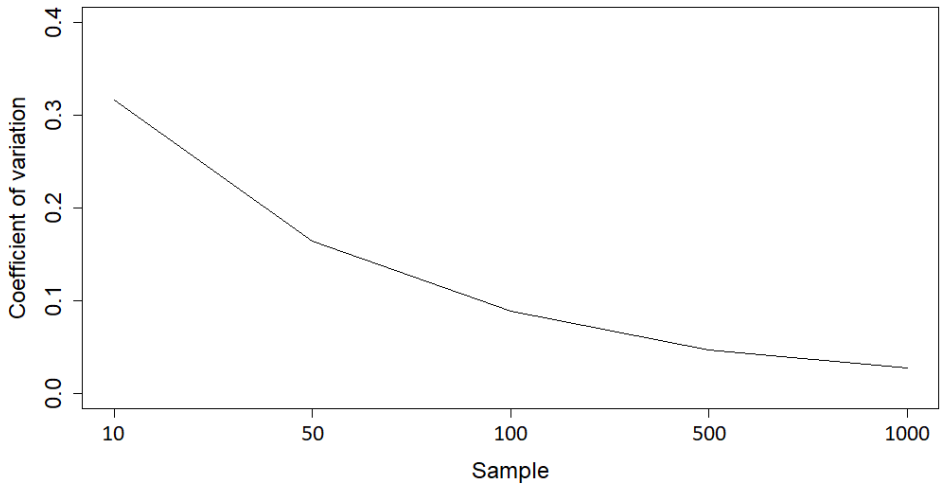
Hero	Deck name	Strategy
Warrior	Tempo Warrior	Hybrid
Warlock	Handlock	Aggressive
Druid	Egg Druid	Defensive
Priest	Dragon Priest	Hybrid
Rogue	Oil Rogue	Hybrid
Mage	Concede Mage	Hybrid
Shaman	BloodLust Shaman	Aggressive
Hunter	Midrange Hunter	Hybrid
Paladin	Aggro Paladin	Aggressive

Source: By the Author.

visibility window from a decision tree, while the BC-Greedy plays by selecting the best action that it can perform at the present considering two States, from a Time Flow, the present and one into the future. Both agents, from Metastone, were evaluated playing against the MCTS and BC-Greedy for each selected hero. To select an adequate quantity of simulations per deck to be performed during the experiments, a coefficient of variation, depicted in Chart 4, of the win rate performance was obtained from a sensitivity analysis of MCTS playing with the deck called Control Warrior against another MCTS playing with the same deck. The sensitivity was measured as an average of 100 executions for each number of simulations equals to 10, 50, 100, 500 and 1000 with a total of 166000 simulations. The total amount of simulation per deck was limited to 1000 since the experiment has an unfeasible computational complexity.

As showed in Chart 4, the coefficient of variation for each number of simulations

Chart 4: Winrate coefficient of variation for Metastone.



Source: By the Author.

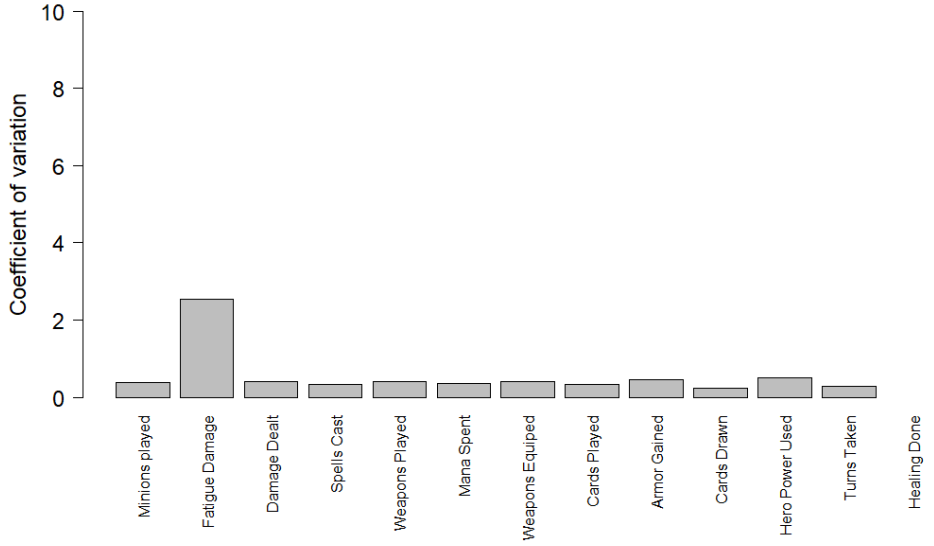
ranges from 0.35 when running 10 simulations to a near stable 0.05 when running 1000 simulations. Despite a reliable range equal to [0.15,0.1] for 50 to 100 simulations, was preferred the usage of 1000 simulations per experiment to obtain a more precise win rate response from *Metastone*. It is important to note that this coefficient of variation is valid for the Warrior deck playing against itself with the MCTS heuristic. Alongside with the win rate analysis, some experiments for T-HearthBot also shows the summarized behavior of the agents over the selected number of simulations. As depicted in Chart 5, the coefficient of variation from 1000 simulations based on the calculated one for the win rate for each extracted attribute statistics, described by the X-Axis, is ranging from 0 to 0.4. Considering the observed coefficient, the total number of simulations for the behavior attributes was maintained into 1000 as accomplished for the win rate analysis.

It is important to note that both coefficients, win rate and behavior attribute, were extracted for the warrior versus warrior MCTS match, thus its is assumed that the statistical fluctuations for all other matches from Metastone will behave accordingly. However, a variation between experiments is expected and should be considered into any taken conclusion.

11.2.1 Metastone agents playing against MCTS

The behavior of the MCTS heuristic, as illustrated in Chart 6, shows the average win rate for each selected hero. Each bar on the X-axis is the mean of the win rate obtained

Chart 5: Attribute behavior coefficient of variation for Metastone.

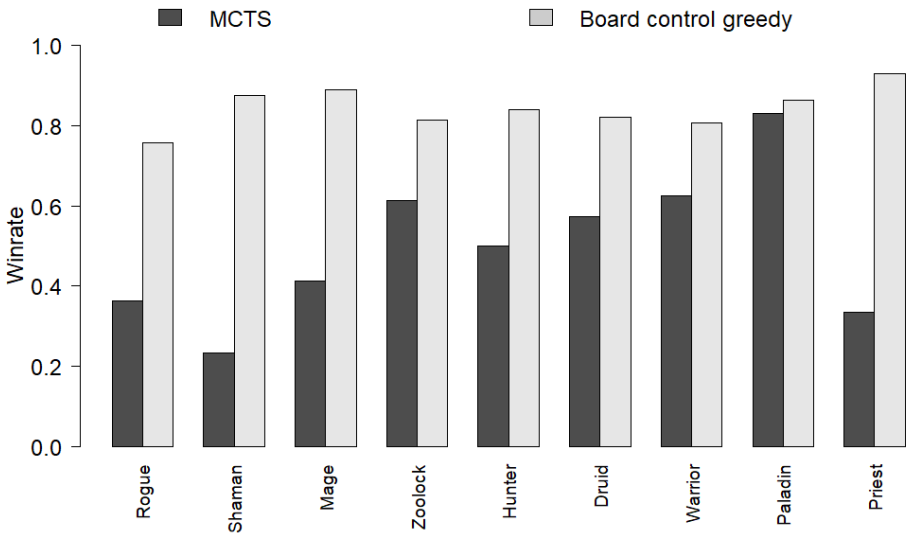


Source: By the Author.

for the indicated hero versus all nine heroes from Table 2. Assuming that the coefficient of variation of the selected number of simulations is low, this graph measures the raw average performance for both selected Metastone main agents, playing against the MCTS heuristic.

As depicted in Chart 6, the MCTS displayed a minimum average near 20% for the Shaman deck against a maximum of 80% for the Paladin. It seems that it can not play well with the Rogue, Shaman, Mage and Priest decks since all of them consists in complex strategies. When playing against itself, MCTS vs MCTS, the expected behavior was that the win rates should be close to 50%, but it did not happen, because some decks strategy has advantages over others as argued in Section 11.2.3. For all the decks there are clear advantages from each one against the others. For example, the Secret Paladin deck appears to be more efficient against all the other decks with its average raw win rate being higher than everyone else. Differently from the MCTS, the BC-Greedy playing against the MCTS displayed an average win rate for all heroes above 75%. The minimum observed win rate for the BC-Greedy was 77% for the Rogue deck, while the maximum observed was near 93% for the Priest deck. This result show that the BC-Greedy is able to perform better, in terms of general average win rate than the MCTS for the selected decks during Metastone simulations.

Chart 6: Selected decks against Metastone Monte Carlo Tree Search.



Source: By the Author.

11.2.2 Metastone agents playing against Board Control Greedy

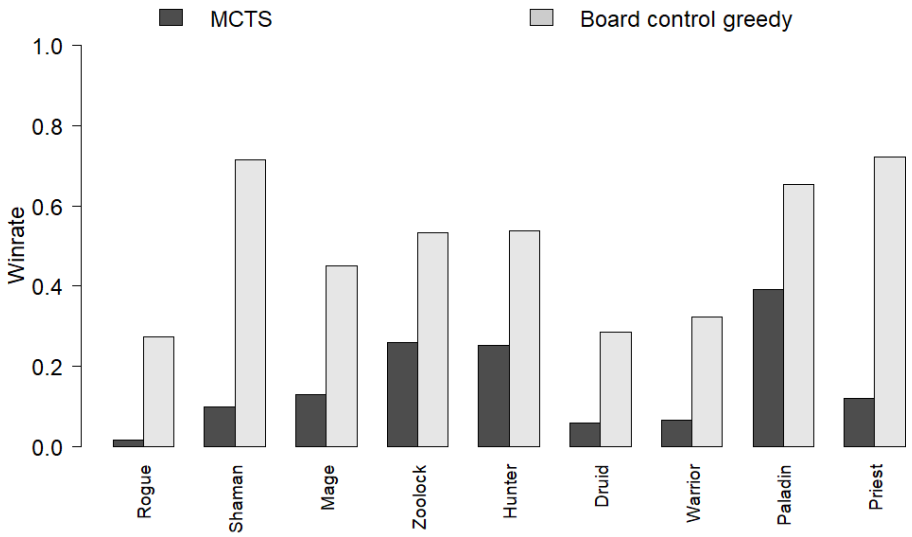
The win rate behavior of the BC-Greedy is showed in Chart 7 and it represents the average win rate, of each selected hero playing against Metastone BC-Greedy, for each selected hero. The bars on the X-axis, from Chart 7, have the same meaning as explained for Chart 6.

As depicted in Chart 7, when playing against the BC-Greedy the average win rate of each deck for each bot has decreased. If analyzing the MCTS, it can be seen that its average win rate for each hero is bellow 45% and in some cases it can be seem bellow 10%. Since the analyzed behavior of the BC-Greedy, playing against the MCTS was better than the MCTS, the presented averages in Chart 7 were expected. On the other hand, the BC-Greedy playing against itself, for each hero, displayed an average win rate performance that ranges from 30% to 75%. Conclusively, for the behavior deployed by Metastone, the BC-Greedy is a better heuristic than the MCTS for the analyzed matches.

11.2.3 Deck win rate signature and discussion

A deck advantage is clearly apparent when evaluating their raw behavior through MCTS, where each deck has its own win rate curve when comparing with all others. For example, in Chart 6 and 7, the average win rate for the Paladin deck is higher than everyone. This behavior is recurrent for each hero, where each one has its own average win rate signature and it occurs because each deck has a specific set of cards that enable

Chart 7: Selected decks against Metastone Board Control Greedy.



Source: By the Author.

players to perform actions that in general subdue part of an enemy's moves. For instance, the Paladin deck is composed by many secrets (i.e. special kind of spells) thus it is possible that those secrets are being used to subdue a partial amount of enemy's moves. It is important to note that for all heroes, its win rate signature is untied to *HearthBot* capacity in subduing moves, but rather on the essence of a deck since a MCTS approach shows the same behavior. All win rate signatures represent how god or bad in average is that deck, thus if considered in a real Hearthstone match it can help players to decide what decks they should use in order to maximize their average win rate.

11.3 HearthBot evaluation

All experiments with *HearthBot* were conducted in a CentOS linux installed in a dual Intel Xeon E5-2620 with the total amount of 24 threads and 64 GB of memory. The proposed ANN was implemented in CUDA/C++ and executed in a Tesla K20 that has 2496 cores and 5 GB of memory. The Graphics Processing Unit memory limits the maximum number of neurons, that the ANN can handle, to 4 million. Moreover, all the feature fields were limited to a maximum of 20 variables to avoid communication bottleneck when executing the learning or prediction processes. The *Metastone* simulator used was in the version 1.2.0 and it was obtained from (DEMILICH, 2016). It encompasses the last season updates to this date from *Hearthstone*. All the simulator routines were implemented with Java by (DEMILICH, 2016), due to this fact the proposed ANN was programmed

to communicate with Metastone through the Java Native Interface (JNI), thus avoiding bottlenecks from interprocess communication.

11.3.1 Experimental design

In this research, *HearthBot* is compared with a MCTS agent available in the *Metastone* simulator in order to evaluate is performance. The BC-Greedy was not used to evaluate HearthBot since when playing against it, HearthBot's average win rate performance for each evaluated hero was very low. This behavior indicates that HearthBot, deployed without temporal learning, may not be able to handle a heuristic like Metastone's BC-Greedy. HearthBot was evaluated in two sets of experiments:

a) On the first set of experiments, called *training*, for each deck, *HearthBot* trains a new neural network playing against the nine decks with the MCTS agent and test it against the same nine decks.

b) On the second set of experiments, called *exploiting*, *HearthBot* was trained with the same decks, but instead it was tested with nine different unobserved decks described by Table 3 from Section 11.2. This enables to verify if *HearthBot*'s associative memory generalizes against new decks.

Moreover, the cognitive code growth and analysis was also made, in Section 11.3.6, during the *training* experiment, to observe the internal behavior of the proposed ANN.

11.3.2 Simulations

During the *learning* experiment, *HearthBot* training was accomplished by playing with each deck against MTCS for all the decks presented on Table 3. This training stage was constituted 1800 simulations per deck and a total of 16200 simulations for the full training. This experiment was performed for all the three configurations in Table 4, what gives a total of 48600 simulations for each deck. When performing, in *learning* experiment, *HearthBot* was evaluated through 1000 games against each trained deck resulting in a total of 243000 simulations considering all the three configurations in Table 4.

The *exploiting* experiment was used to verify its performance against unknown opponents. During this experiment, *HearthBot* uses all the trained decks to play against every unobserved deck described by Table 3. The win rate performance was extracted as

in the *training* evaluation, with 1000 simulations each in a total of 81000 simulations. This experiment was also performed using the configuration *reso-90* for a high precision and some degree of generalization.

The bot performance was evaluated through the win rate obtained from the simulations for each experiment. The win rate of a deck was measured by calculating the ratio of wins in relation to the total number of games evaluated. All the win rate performance was finally evaluated with the geometric mean for all the experiments, thus avoiding over-evaluation from different decks for each experiment.

11.3.3 Parameters choice

In the literature, a multi-channel ANN is typically used with a high resonance criterion, above 80%, to provide some degree of specialization. For HearthBot, three configurations were used, they are specified in Table 4. The configuration *reso-80* has 80% resonance in order to obtain some degree of specialization, in contrast, configuration *reso-90* was created with a neuron matching above 90%, and finally, the configuration *reso-90+* was used for near precise matching with a vigilance criteria above 95%. The last variable of each parameter set were equal to 0 for the reason that the reward field is shared between cognitive codes from the same category, thus categorizing concisely. Finally, all the learning rates were configured equally to the resonance of each configuration for fast learning.

Table 4 – Parameters choices for each test from *training* and *exploiting* experiments.

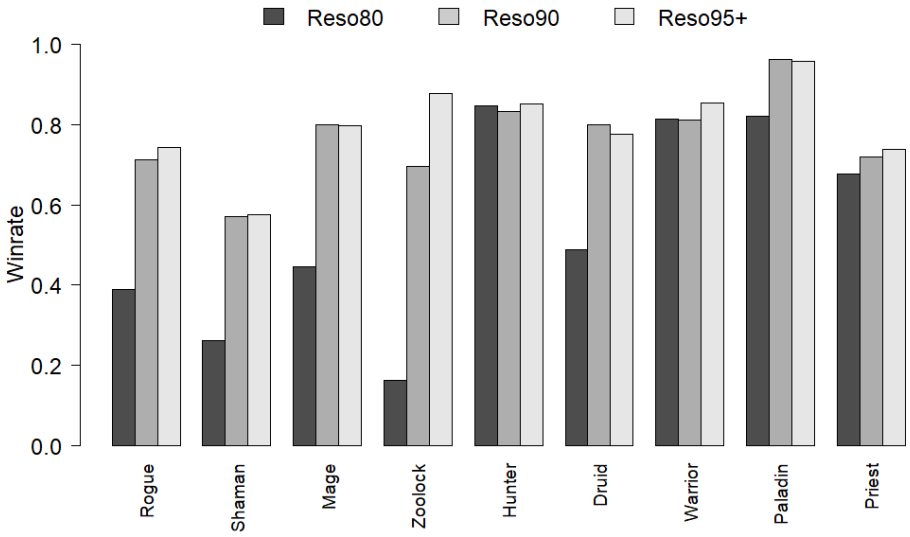
reso-80			*reso-90*			*reso-90+*		
$\rho^1 = 0.8$	$\gamma^1 = 1$	$\beta^1 = 0.8$	$\rho^1 = 0.9$	$\gamma^1 = 1$	$\beta^1 = 0.9$	$\rho^1 = 0.9$	$\gamma^1 = 1$	$\beta^1 = 0.9$
$\rho^2 = 0.8$	$\gamma^2 = 1$	$\beta^2 = 0.8$	$\rho^2 = 0.9$	$\gamma^2 = 1$	$\beta^2 = 0.9$	$\rho^2 = 0.95$	$\gamma^2 = 1$	$\beta^2 = 0.95$
$\rho^3 = 0.8$	$\gamma^3 = 1$	$\beta^3 = 0.8$	$\rho^3 = 0.9$	$\gamma^3 = 1$	$\beta^3 = 0.9$	$\rho^3 = 0.95$	$\gamma^3 = 1$	$\beta^3 = 0.95$
$\rho^4 = 0.8$	$\gamma^4 = 1$	$\beta^4 = 0.8$	$\rho^4 = 0.9$	$\gamma^4 = 1$	$\beta^4 = 0.9$	$\rho^4 = 0.98$	$\gamma^4 = 1$	$\beta^4 = 0.98$
$\rho^5 = 1.0$	$\gamma^5 = 1$	$\beta^5 = 1.0$	$\rho^5 = 1.0$	$\gamma^5 = 1$	$\beta^5 = 1.0$	$\rho^5 = 1.0$	$\gamma^5 = 1$	$\beta^5 = 1.0$
$\rho^6 = 0.0$	$\gamma^6 = 0$	$\beta^6 = 0.8$	$\rho^6 = 0.0$	$\gamma^6 = 0$	$\beta^6 = 0.9$	$\rho^6 = 0.0$	$\gamma^6 = 0$	$\beta^6 = 0.9$

Source: By the Author.

11.3.4 Overall performance and discussion

HearthBot overall performance during the *training* experiment is illustrated in Chart 8. This graph represents the geometric tendency from the raw performance, and it was obtained during the *training* experiment. As depicted in Chart 8, the MCTS wins against the random approach with an average of 9% win rate difference. This result suggests that the MCTS, as provided by *Metastone*, is not a good approach to play this game since its performance is slightly better than a random one. Another explanation is that the solution search space is so big that the MCTS could not find a reliable solution with its heuristic.

Chart 8: HearthBot win rate against all the decks of the experiments from the *training* experiment.



Source: By the Author.

The results showed in Chart 8 also shows that *HearthBot* overcomes both, random and the MCTS approaches for the configurations *reso-90* and *reso-90+*. In contrast, configuration *reso-80* lacks the necessary precision to do good predictions, since its overall performance is poorer compared to *reso-90* and *reso-90+*. For instance, the deck used with the Warlock has the worst win rate, near 1% and is below the random approach. This behavior could be explained as a generalization in such a level that *HearthBot* could not differentiate distinct deck cards from each other, thus resulting in learning moves that lead to a negative reward. In contrast, the Warrior and the Hunter have a win rate tendency near 80% for the configuration *reso-80*, what implies in all the moves receiving almost the same reward due to the play style of the decks or due to a great similarity between all

cards on those decks. For the other decks, the configuration *reso-80* performed equally or slightly worse than the MCTS approach, which shows that the MCTS, in overall, could be easily defeated by *HearthBot*.

HearthBot received a great improvement when using the configuration *reso-90*, as depicted in Chart 8. This is probably because of its enhanced precision obtained by a $\rho > 0.9$, leading to more precise matchings between context and cognitive codes. Also, the configuration *reso-90* uses more cognitive codes, to represent more distinct information obtained from the environment. This representation enhancement can be seen directly into the Warlock performance that is slightly higher than both, random and MCTS approaches. For all the other heroes, the configuration *reso-90* performs well and overcomes the MCTS by an average of 31% of win rate difference. In comparison to other heroes, the Druid performed the best and scored an average of 78% win rate.

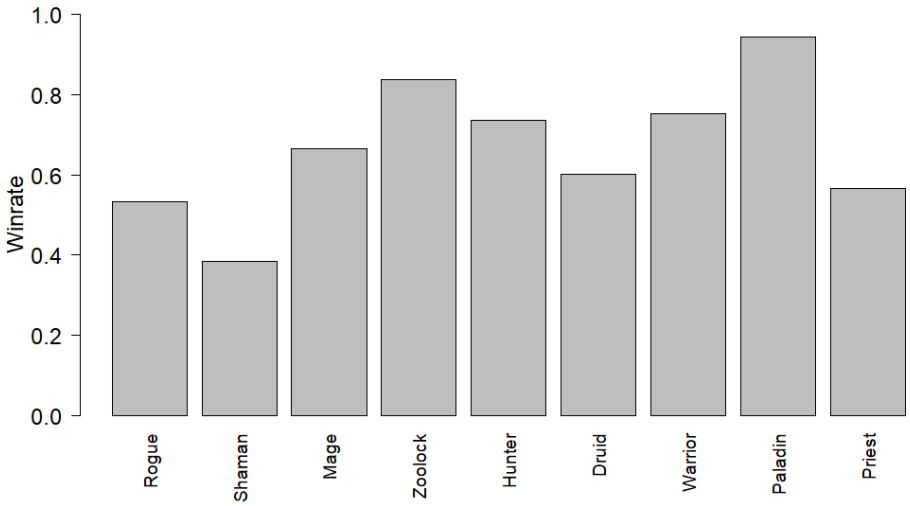
In contrast to *reso-80* and *reso-90*, configuration *reso-90+* performed similarly to *reso-90*, but with more cognitive codes and a more precise matching with its $\rho > 0.95$ for some fields. This impacts into the Warlock hero, that have almost the higher win rate near 90%. This configuration performed also better for the Warrior, Priest, Mage, Shaman and Hunter, thus being in overall the most efficient one in terms of win rate performance. The Druid hero gets bellow from its counterpart in configuration *reso-90*, that could be explained as outliers inside the geometric tendency since the coefficient of variation is higher than 0. The major problem with this configuration is the training time that relies on the number of neurons used.

11.3.5 Overall performance against unobserved decks

Overall, as illustrated in Chart 9, the geometric tendency was 21% greater in favor to *HearthBot* compared to MCTS. The lowest win rate was from the Control Warrior, with an average win rate performance near 73% and a maximum as 93% from Secret Paladin. These results show the capability of the proposed ANN to generalize while maintaining the integrity of the learned data. Besides the good results from heroes with a clear advantage against others, *HearthBot* performed regularly. For instance, the Aggro Shaman win rate performance was 48%, and this means that it loses more than wins against the unknown opponents, nevertheless, this was the win rate performance below 50%.

HearthBot performed better than the MCTS, but it is clear that the difference

Chart 9: HearthBot win rate against unknown decks.



Source: By the Author.

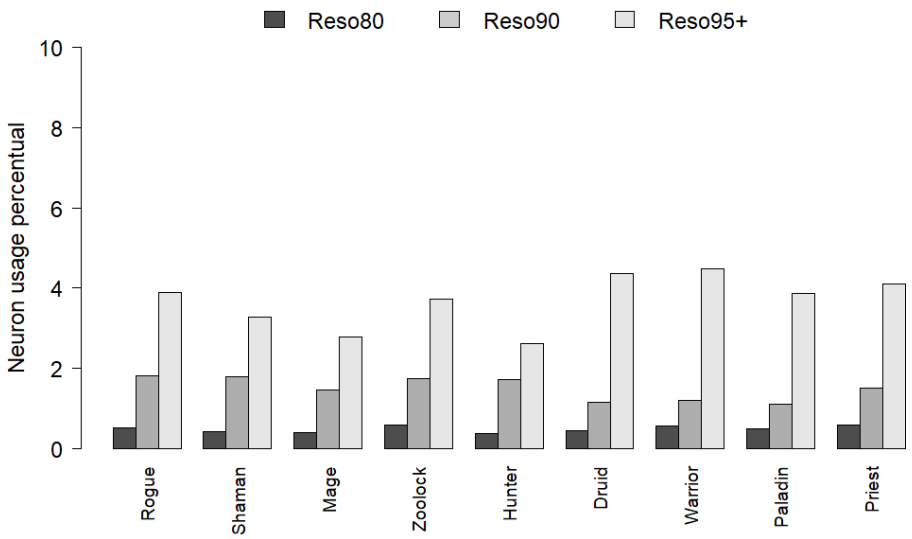
of 21% in average was not higher than the difference obtained against trained decks. Furthermore, the overall performance for all decks was proportional to the ones obtained on the *learning* experiment, concluding that *HearthBot* seems better against the MCTS for trained and unobserved decks.

11.3.6 Cognitive code analysis

The proposed ANN displays a neuron usage, calculated as the total number of neurons used to codify all the training data, depicted in Chart 10, proportional to the precision described by the three configurations *reso-80*, *reso-90* and *reso-90+*. For instance the lowest values of usage, bellow 0.5%, were from the Druid, Mage, Shaman, Hunter and Paladin from *reso-80*.

The observed behavior happened due to the fact that decks have cards that are considered similar by the ANN, what may achieve a better context generalization per codified neuron. For the configuration *reso-90*, it is possible to see that the Warrior, Warlock, Priest and Rogue obtained an usage above 1.5%, indicating not that the effects have a higher resonance criteria, but also that for those heroes the *Hearthstone* contexts were less generalized. The configuration *reso-90+* was the most expensive, it used almost 5% of the total capacity. Despite 5% seems to be a low value, if the hardware used to perform all the operations of the proposed ANN had a considerable lower memory capacity it could be practically impossible to use it.

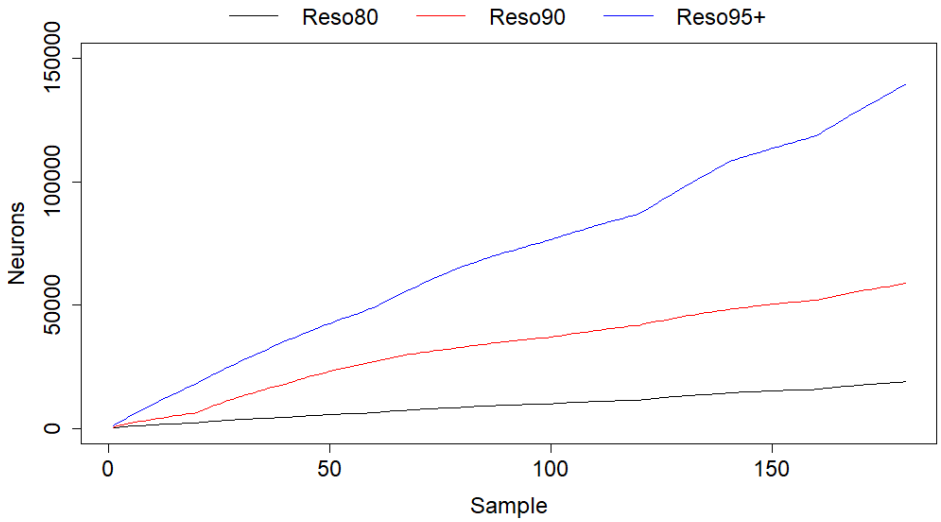
Chart 10: Neurons usage represented by the Y-axis as the total amount of the ANN capacity.



Source: By the Author.

Neuron growth can be a problem if it happens too fast because it could easily get beyond the ANN limits, for *HearthBot* it happens in a quasi linear manner as showed in Chart 11. For the configuration *reso-80*, *HearthBot* uses lower quantities, bellow 20000, this happens due to the lower resonance criteria. In contrast, the configuration *reso-90* uses lower values at the beginning and higher values above 20000 for the rest of the training stage. It can also be observed that the behavior of the configuration *reso-90* is almost logarithmic at the beginning and starts to become linear after 50% of the execution time.

Chart 11: Neuron growth geometric tendency for each trained deck.



Source: By the Author.

The observed behavior, from Chart 11, could be explained as a behavior of *Metastone* in relation to the randomness for each simulation, otherwise this growth would be almost linear. Configuration *reso-90+* has the largest growth rate, that exceeds 100000 neurons when reaches 70% of the training stage. This growth behavior is explained also by the resonance criteria for each configuration, which has a trade-off between more precision and code size or more generalization and less precision in environment representation. These results show that the proposed architecture could be used with a commercial Graphics Processing Unit without memory limitations instead of ones manufactured for the scientific domain as the Tesla K20.

11.4 T-HearthBots evaluation

The *HearthBot* evaluation was performed to verify its ability to play Hearthstone. By contrast, the proposed T-HearthBot variants described in Section 11.4.1 were evaluated to investigate how the proposed creative HearthBot, deployed with the HARP and UAM-ANN, perform on Metastone using the proposed Hearthstone action coding models, behavioral and spectrum. Differently from HearthBot, all T-HearthBots were also evaluated playing against the BC-Greedy since it is expected that their behavior against it should be better with the deployed Q-learning model. The experiments for all T-HearthBots variants were conducted in Windows 10 Pro installed on a core I7-4700Q with the total amount of 4 threads and 4 physical cores with 8 GB of memory. All the proposed T-HearthBots were implemented directly with Java and were executed on the Central Processing Unit, since the temporal models as presented by (TAN et al., 2008) spend less neurons than a memory based model. The *Metastone* simulator used was in the version 1.2.0, as for HearthBot, and it was obtained from (DEMILICH, 2016).

11.4.1 Experimental design

Five T-HearthBot variants were evaluated in order to see how the HARP and UAM behave under a simulated Hearthstone environment. The first bot is called T-HearthBot, proposed in Chapter 10, that incorporates the Q-Learning FALCON architecture with the proposed Hearthstone model. The second evaluated bot is called TR-HearthBot, where it is the T-HearthBot deployed with the reactive model and a behavioral Hearthstone action coding, that uses an action mask described in Section 6. On the other hand, the creative T-HearthBot called CTH-HearthBot, was deployed with the proposed HARP. In addition, the TU-HearthBot, a composition of the proposed UAM with T-HearthBot, was also evaluated. Finally, the creative T-HearthBot, called CTUH-HearthBot, based on a HARP and UAM-ANN, was evaluated. The evaluation of all T-HearthBot variants were organized in 5 main experiments described as follows.

a) **Winrate convergence:** This experiment shows how good the convergence was for the proposed T-HearthBots when playing against two Metastone heuristics, the MCTS and the BC-Greedy. In this experiment, all T-HearthBots play against 2 selected decks and their behaviors are analyzed through observed epochs. The deck

selection, for this experiment, was based on the worst observed performance obtained for HearthBot. The deck selection process is described in detail in this section and further complemented in Appendix A. This experiment is presented by Sections 11.4.4 and 11.4.4.

b) **Search space exploration:** The cognitive code growth of the creative system was evaluated for the Shaman CTH-HearthBot versus the MCTS Paladin in a controlled environment. The evaluation permits to see how the creative Hearthbot can better explore the search space when seeking for new actions to perform. All analysis from this experiment are presented in Section 11.4.6

c) **General win rate:** This experiment is responsible to evaluate the average win rate of CTUH-HearthBot, one of the best evaluated proposals for T-HearthBots in terms of cognitive code growth and obtained performance, for each playable hero against every one in Table 2. This experiment performs a similar evaluation as performed for HearthBot in Section 11.3.4 and it is presented in Section 11.4.7.

d) **Overall performance against unobserved decks:** This experiment is responsible to evaluate the average win rate of CTUH-HearthBot, but instead it is tested with nine different unobserved decks described in Table 3 from Section 11.2. This experiment performs a similar evaluation as performed for HearthBot in Section 11.3.5 and it is presented in Section 11.4.8.

e) **Cognitive code:** As conducted for HearthBot, a cognitive code growth is also presented at the last section for T-HearthBots evaluation, in order to investigate how the FALCON behave. The evaluation aims to investigate how much storage space is expended by T-HearthBots, when deployed as an ARAM and the proposed UAM-ANN.

Differently from *HearthBot*, all *T-HearthBots* win rate evaluations were also accomplished by observing their convergence during epochs in order to observe the temporal learning behavior over time. All T-HearthBots were evaluated playing against MCTS agents and since they were assembled based on temporal learning techniques, then it is expected that they behave in a better way than HearthBot. Based on that assumption, all T-HearthBots were also evaluated playing against the Board Control Greedy heuristic from Metastone. That stems from the fact that, the Board Control Greedy bot shows a better

performance than the MCTS as presented in Section 11.2. Moreover, to evaluate their convergence, individually, 2 heroes were selected, Hunter and Warrior, from *HearthBot's* exploring experiment table, where the Hunter played against eh MCTS and the Warrior played against the BC-Greedy. The selected heroes represent two of the worst cases, lowest *HearthBot* performance in terms of win rate, observed during *HearthBot's* performance.

In order to select which enemy for each selected deck, the Hunter and Warrior were evaluated playing against everyone from Table 3. Through all analyzed data obtained from the individual win rate performance of the Hunter deck, where the enemy was the MCTS heuristic, could be observed that Metastone's bots display the worst performance when playing against the Paladin deck. Furthermore, when playing against a BC-Greedy heuristic, the Warrior deck displayed the worst performance against the Shaman. Conclusively, the selective enemies, to conduct the win rate convergence experiment, were the Paladin and Shaman deck. The behavior of the Hunter deck is presented in details on Appendix A. In addition, complementary behavior analysis for other decks that performed worst during HearthStone evaluation is also presented in Appendix A.

11.4.2 Simulations

According to the Metastone behavior described in Section 11.2, a total amount of 1000 simulations is necessary in order to reduce the coefficient of variation of the observed winrates, to a value near 0, that satisfactorily reduces statistical fluctuations displayed by the simulator. Considering that assumption about the coefficient of variation, the number of simulations for each main experiment described in Section 11.4.1 is given as follows:

a) The winrate convergence experiment, as described in Section 11.4.1, was conducted to measure the win rate convergence in a determined number of epochs. To permit collecting a reasonable amount of epochs, 5000 simulations are performed for each win rate convergence test. For the total amount of simulations defined, 200 epochs were measured per experiment, where each epoch is composed by 25 simulations.

b) Data from the search space exploration experiment, was collected during 5000 simulations obtained in 200 epochs. Each epoch was composed by 25 simulations as accomplished for experiment winrate convergence experiment.

c) For the general win rate experiment, 9000 simulations were conducted in order

to collect the general average for each hero. To enable all evaluated bots to learn strategies properly, 100 simulations are performed earlier until reaching a epsilon equal 0 for each match. The average for one hero is collected during 1000 remaining simulations giving the total of 9000 per hero and HearthBot.

d) For the overall performance against unobserved deck experiment, each deck was trained with 200 simulations playing against everyone from Table 2. In order to verify its performance against unobserved data, 1000 simulations were performance when playing against each deck from Table 3, thus giving a total of 1800 simulations for training and 9000 simulations for evaluation. For instance, this is the same experiment as conducted for HearthBot.

e) In the cognitive code experiment, the cognitive code growth was measure through 12500 simulations in order to capture its behavior in a larger observation window. It was also divided into 500 epochs, where each epoch is composed by 25 simulations.

11.4.3 Parameters choice

All evaluated T-HearthBots are based on a reactive or Q-Learning techniques as described in Section 6, thus all their dynamics are guided by the parameters described in Table 5. The erosion rate parameter, configured as 0.2, is responsible to determine a slow erosion rate for the TR-HearthBot. The reinforcement rate was configured as 0.5 for a midrange reinforcement rate when performing with the TR-HearthBot. Furthermore, the decay rate was configured as 0.0005 for a slow decay rate, thus the TR-HearthBot retains learned information longer. The pruning threshold for the TR-HearthBot was configured to 100 neurons, thus it will throw away irrelevant information earlier during the learning process and performing process. For all T-HearthBots, the epsilon decay, configured as 0.005, seeks for a fast exploration to exploit often. The initial epsilon starts at 0.5, thus reinforcing the ability to exploit often, since it is a low value for a decay equals to 0.005. When using the Q-Learning, the discount parameter was configured as 0.1 and the learning rate as 0.5 for a midrange learning speed.

The T-HearthBots parameters were configured as shown by Table 6, where the env field 1 and env field 2 are representing the proposed environment coding for all T-HearthBots presented in Chapter 10. To achieve fast learning, all learning rates, for all fields, were configured as 1. For the T-HearthBot and CTH-HearthBot, the learning

Table 5 – General parameters for all T-HearthBot variants

General parameter	Value
Erosion rate	0.2
Reinforcement rate	0.5
Decay rate	0.0005
Prunning threshold	100
Vigilance reinforcement rate	0.001
Epsilon decay	0.005
Initial Epsilon	0.5
QDiscount	0.1
QLearning rate	0.5

Source: By the Author.

vigilances on the environment fields 1 and 2 were configured as 0.8 to achieve some degree of generalization. For the HARP system, the Expectation ART, in CTH-HearthBot and CTUH-HearthBot, it uses the same learning vigilances to categorize environments in fields 1 and 2. Differently, the learning vigilance for the TU-HearthBot and CTUH-HearthBot in channel 1 is configured as 1, since the expected number of possible actions is lower than the total expected amount of environments. However in channel 2, both uses the learning vigilances, for field 1 and 2. The gammas and alphas variables are configured as 1 or 0.1, for each respective attribute represented by its column. Furthermore, the adaptive vigilance method is used by all networks that codes environment fields 1 and 2. The fuzzy readout parameter is never used, because it can cause false predictions. Finally, the direct reward is always used inside all Q-Learning variants of the T-HearthBots.

Table 6 – Field parameters for all T-HearthBots, HARP and UAM proposals

General parameter	Env field1	Env field2	Action field	Reward
learning rate	1	1	1	1
learning vigilances	0.8	0.8	1	0
gammas	1	1	1	0
alphas	0.1	0.1	0.1	0.1
adaptive vigilances	true	true	false	false
resonance to predict	false	false	false	false
fuzzy readout	false	false	false	false
use direct reward	true			

Source: By the Author.

Neuron composite activation function selects which operation will be used to perform the neuron activation and resonance checking procedures. In this research, T-HearthBot

uses the ART I operation for all its fields to represent the original ARAM FALCON architecture proposed by (TAN, 2004). TU-HearthBot and CTUH-HearthBot use ART I operations for the env field 1 and env field 2, from channel Y, to generalize an environment, ART II for learning and prediction in channel 1, that represent actions and the proximity activation in resonance checking for action learning. This configuration enables to learn and generalize environments and precisely categorize binary patterns when performing action learning and prediction. The CTH-HearthBot uses a similar configuration, as TU-HearthBot and CTUH-HearthBot, for its fields as showed in Table 7. With respect to all HARP based networks, ART I operations are used inside their env 1 and env 2 fields for environment generalization during Bayesian surprise vector categorization.

Table 7 – Neuron activation parameters for all T-HearthBots

Bot	ART I	ART II	Proximity
T-HearthBot	All fields	-	-
TR-HearthBot	All fields	-	-
TU-HearthBot	Env1 and Env2 in channel Y	Learning in action channel 1 and prediction in channel Y	Resonance in action channel 1
CTH-HearthBot	Env1 and Env2	Learning in action field and prediction	Resonance in action field
CTUH-HearthBot	Env1 and Env2 in channel Y	Learning in action channel 1 and prediction in channel Y	Resonance in action channel 1

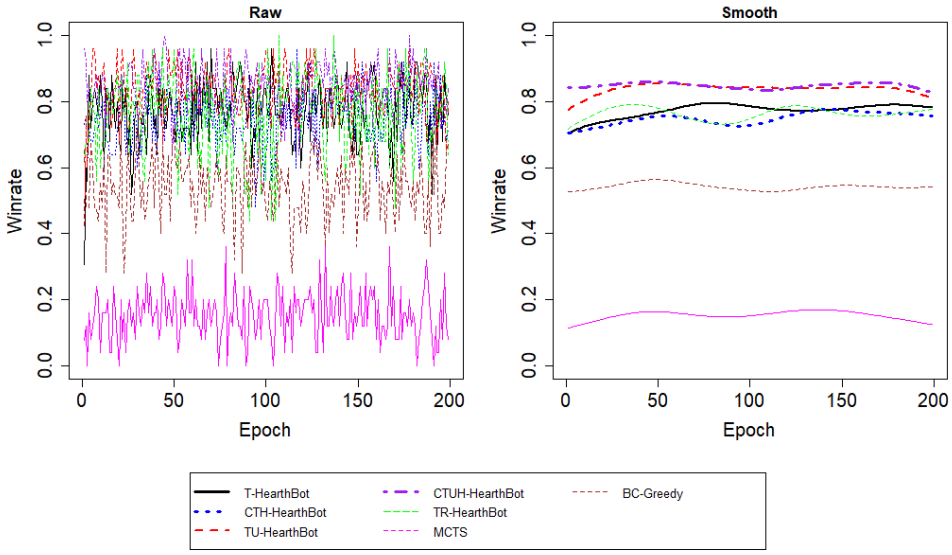
Source: By the Author.

11.4.4 Behavioral T-HearthBots winrate

The win rate convergence for all T-HearthBots, assembled using a behavioral action model, was evaluated for all the selected heroes described in Section 11.4.1. The selected Hunter deck relies in a face strategy, where the best moves for this deck are related in dealing direct damage to the enemy hero. On the win rate convergence analysis for this deck, depicted in Chart 12, the TU-HearthBot and CTUH-HearthBot received an average win rate near 80%, by contrast, the T-HearthBot, CTH-HearthBot and TR-HearthBot received an average win rate performance near 70%. Moreover, the observed performance for the MCTS was near 15% and for the BC-Greedy was near 53%.

The behavior of the Hunter deck playing against the MCTS Paladin, as illustrated

Chart 12: Winrate convergence for Face Hunter versus Metastone MCTS playing with Secret Paladin.

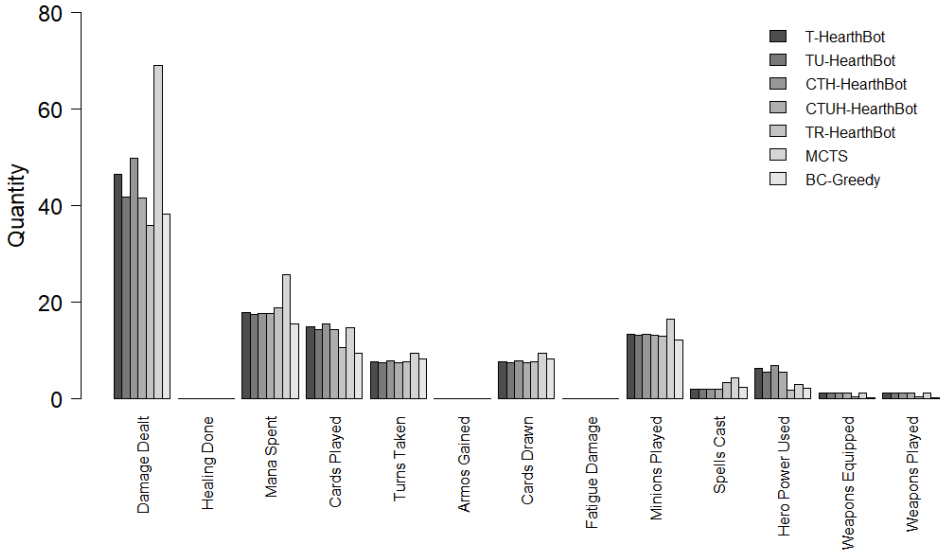


Source: By the Author.

in Chart 13, shows that the MCTS tends to spend more mana by playing more minions and spells. The lowest behavior of the MCTS can be explained as the lack of ability of the bot in deciding efficiently how to expend those resources. Furthermore, as already observed from the previously presented behaviors, the damage dealt does not correlate directly with the win rate performance, since both, T-HearthBot variants and the BC-Greedy, had displayed a low damage dealt, besides that, all T-HearthBots displayed better performance than the BC-Greedy. In addition, it seems that wisely deciding how to spend mana can also influence how good the bot will behave. If analyzing the amount of minions played, can be noted that the MCTS had played the most and for all other bots the quantity of played minions stayed similar. The best performance bot, obtained for the creative CTUH-HearthBot, with the fastest convergence for the Hunter experiment, had displayed a higher spell usage with a low damage dealt, same mana spent as other HearthBots and played the least amount of cards. This behavior could indicate that the converged simulated POMDP, obtained through the UAM with 100% resonance in channel 1 for the Hunter deck, tends to use more spells to dominate the battlefield in order to exploit better its face capabilities.

The win rate convergence for the Warrior versus BC-Greedy Shaman is depicted in Chart 14. For this experiment, all T-HearthBot variant, excluding T-HearthBot, received an average win rate performance near 60% after their convergence. It is important to note that in this experiment, the convergence of T-HearthBot and CTH-HearthBot follow a

Chart 13: Behavior statistics for Face Hunter versus Metastone MCTS playing with Secret Paladin.

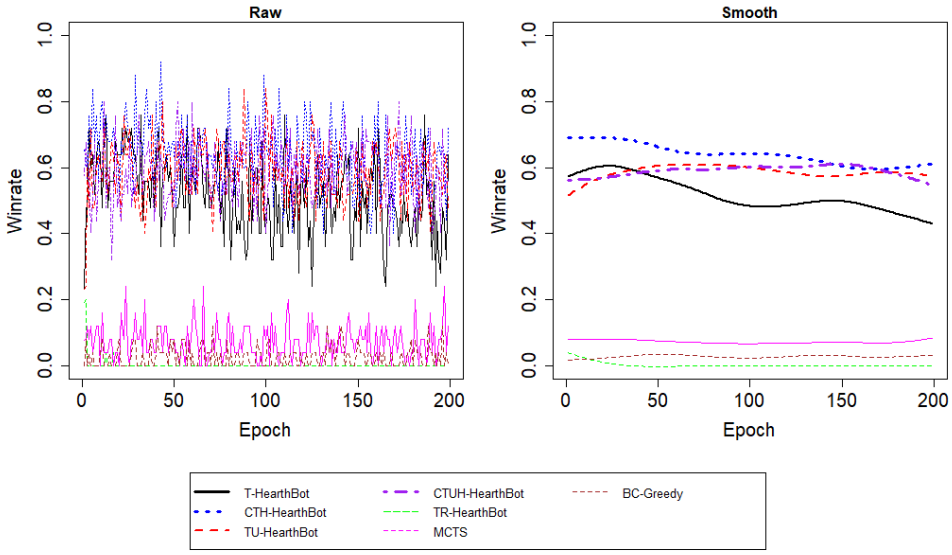


Source: By the Author.

negative slope. This behavior can be explained as the lack of ability from those bots in learn Q-values correctly, since both possess the highest number of neurons, because they are not exploiting a tree structure as the TU-HearthBot and CTUH-HearthBot. For this experiment, the TR-HearthBot received a win rate performance, equal to 1%, that is bellow the MCTS. Since the defensive Warrior deck relies mostly in building up the battlefield in order to avoid damage, it seems the TR-HearthBot was not able to optimize the virtual POMDP considering the future, thus it had not exploited well the deck strategy. The MCTS received a win rate performance near 5% and the BC-Greedy received a win rate performance near 10%. Since both are handcrafted methods, this behavior was expected when playing against the BC-Greedy heuristic from Metastone.

The behavior of the TR-HearthBot, as showed in Chart 15, received the lowest counting for all observed attributes on the X-Axis. This reflects, explicitly, on its win rate performance when playing the game, as depicted in Chart 14. For all other bots, the counted behavior attributes follow the same pattern, where all T-HearthBots were observed with the same counting and the MCTS and BC-Greedy with relative lower ones. In addition, the Warrior deck strategy, that relies mostly in defending the battlefield and hero with armor skills and defensive minions, reflects directly on the observed armor gained, and minions played that received relatively higher counting if compared to the other decks. It was also observed that this deck permits players in complementing their battlefield by using weapons, thus a counting for weapon usage is also present on its

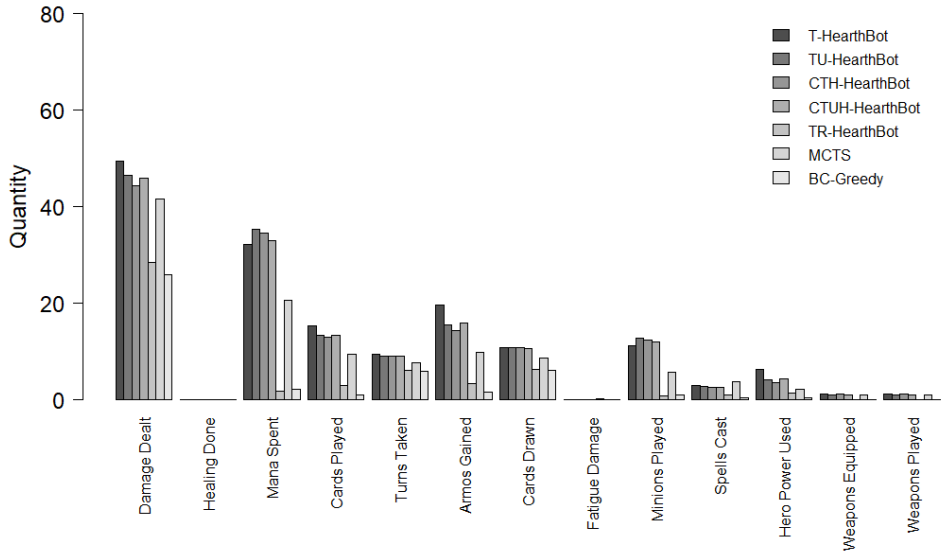
Chart 14: Winrate convergence for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

behavior counting illustrated in Chart 15.

Chart 15: Behavior statistics for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



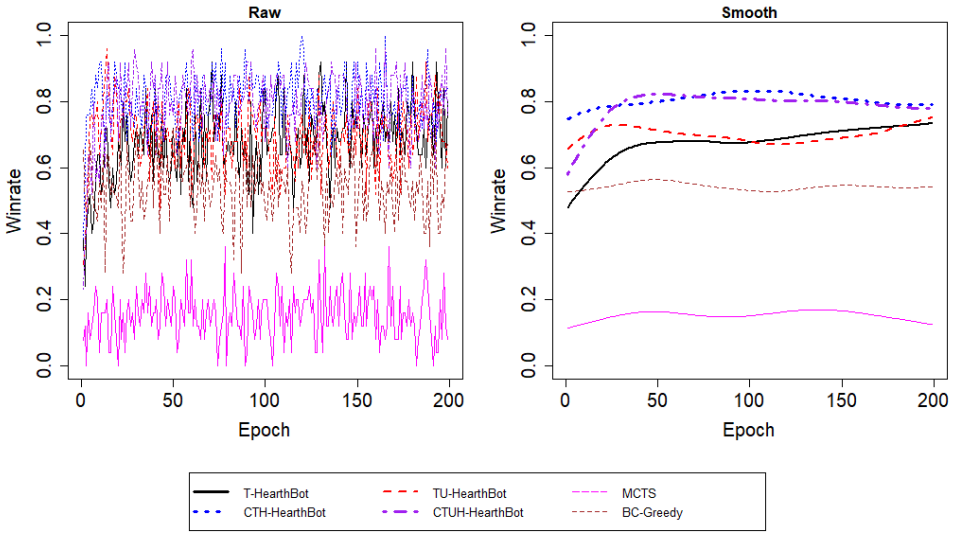
Source: By the Author.

11.4.5 Spectrum T-HearthBots winrate

When analyzing the win rate convergence of the Hunter versus MCTS Paladin, depicted in Chart 16, it can be noted that the creative bots obtained the best performance again, where the CTH-HearthBot and CTUH-HearthBot obtained an average win rate

performance of 80%. The TR-HearthBot and T-HearthBot obtained an average win rate performance of 68%. Furthermore, all T-HearthBot variants received a higher win rate than all analyzed Metastone agents. For instance, the BC-Greedy received an average win rate of 56% and the MCTS received an average win rate of 17%, which indicates a performance gain higher than 50%, if comparing all T-HearthBots with the MCTS, and a performance gain higher than 20%, if comparing with the BC-Greedy.

Chart 16: Winrate convergence for Face Hunter versus Metastone MCTS playing with Secret Paladin.

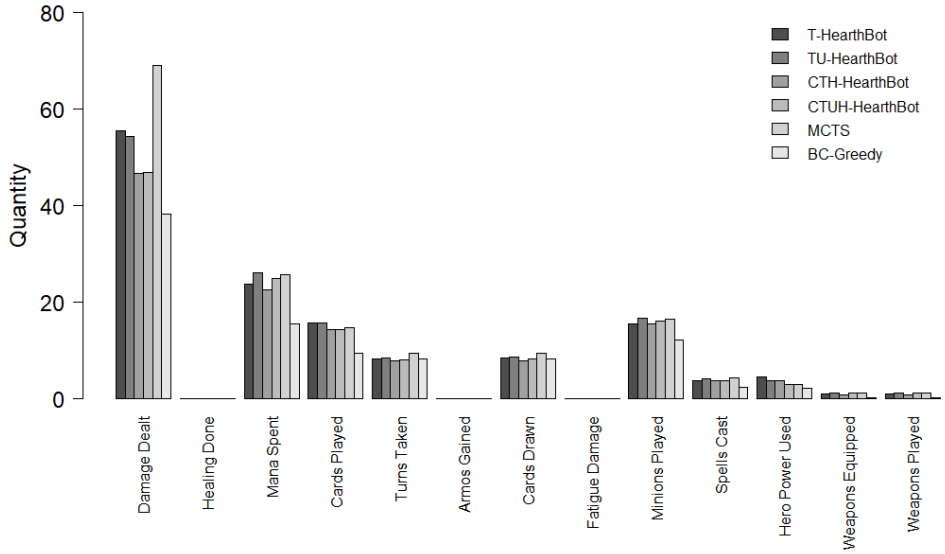


Source: By the Author.

The Hunter deck displays a behavior similar to the one observed for the Mage deck, as showed in Chart 17. All the creative HearthBots, CTH-HearthBot and CTUH-HearthBot, displayed a lower attribute counting for the damage dealt, cards played and cards drawn. Those counted attributes represent the ones responsible in guiding the deck's strategy, where managing those will possibly increase an agent's performance. As also depicted in Chart 17, the attribute that exhibits the highest counting was the damage dealt, for the BC-Greedy. It could indicate that the BC-Greedy heuristic, as discussed on the previous presented behavior analysis, tries to give the highest amount of damage as possible without caring about how much mana it spent.

The win rate convergence for the Warrior versus BC-Greedy Shaman, illustrated in Chart 18, shows that all creative HearthBots displayed an average performance of 50%. In addition, the TU-HearthBot achieve a similar average performance, of 44%, than its creative counterparts. Furthermore, it seems that the TU-HearthBot scored a lower average

Chart 17: Behavior statistics for Face Hunter versus Metastone BC-Greedy playing with Secret Paladin.



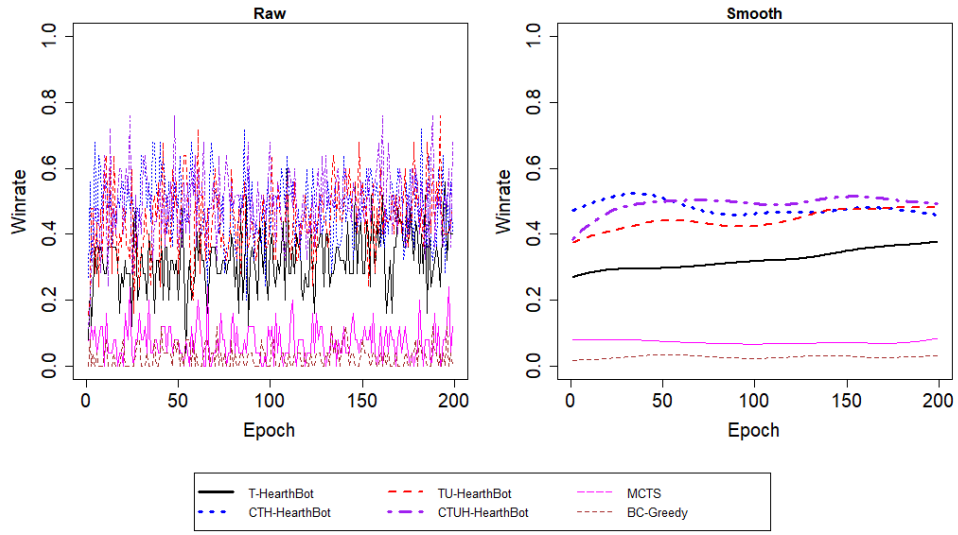
Source: By the Author.

win rate since it did not explore well during the exploration phase before converging. On the other hand, the T-HearthBot achieved an average win rate of 30%, what leads to an average win rate gain, for the creative HearthBots, of 20%. When analyzing the MCTS and BC-Greedy, it can be seen that their performance where considerable worst than all the proposed T-HearthBots, where the MCTS achieved an average win rate of 3% and the BC-Greedy achieved an average of 9%. Since the main strategy of the Warrior is on defending its hero through defensive minions and spells, it was expected to observe that performance for all the handcrafted bots from Metastone that can not deal well with temporal information.

As depicted in Chart 19, the Warrior versus BC-Greedy Shaman displayed a behavior similar to the one observed for the Rogue versus BC-Greedy Shaman. However, the presented behavior in Chart 19 does not show discrepancies that can be interpreted as a sign of good or bad performance. All attributes were quite similar for all T-HearthBot, but it can be seen that the T-HearthBot has spent less mana and it also played fewer minions. This behavior can explain its poor performance in comparison to other T-HearthBots although all other T-HearthBots played nearly identically. The MCTS and BC-Greedy displayed in this experiment the lowest performance, as showed in Chart 19 and this behavior seems to happen recurrently.

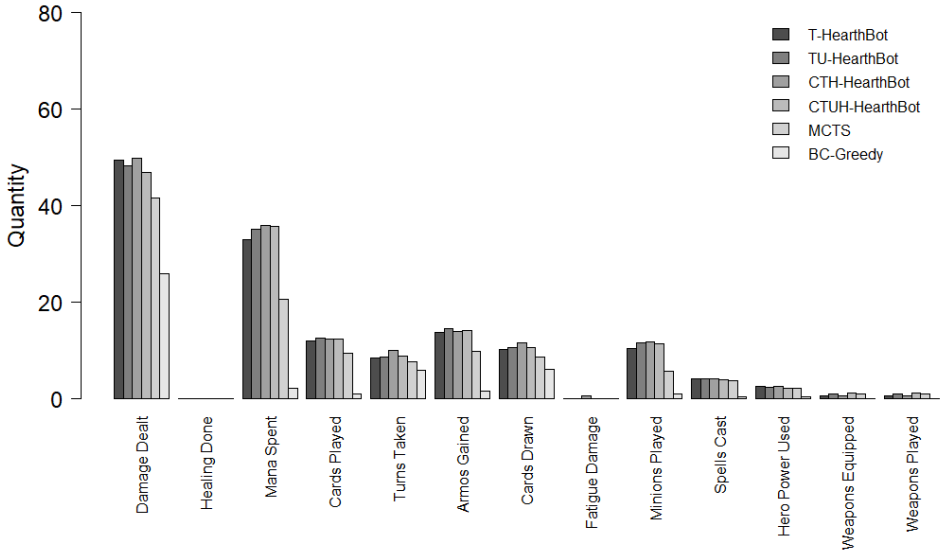
The win rate convergence tests, from Sections 11.4.4 and 11.4.5, describe the behavior of T-HearthBots during its performance. All T-HearthBots obtained a similar

Chart 18: Winrate convergence for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

Chart 19: Behavior statistics for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

performance, even considering that the observed win rate for the creative HearthBots were higher for the presented matches. Moreover, all T-HearthBots performed better than the handcrafted approaches, MCTS and BC-Greedy, demonstrating the temporal technique can win even against the BC-Greedy in terms of average performance. It is important to note that the conducted experiments were based on two matches, the Hunter versus Paladin and Warrior versus Shaman, thus they do not represent well the behavior of the agent when playing with other types of decks. Nevertheless, in order to complement the conducted experiments with a more diverse analysis it was repeated with 6 of the worst performance decks obtained from the HearthBot performance analysis. The aforementioned research is presented at Appendix A.

11.4.6 HARP search space exploration

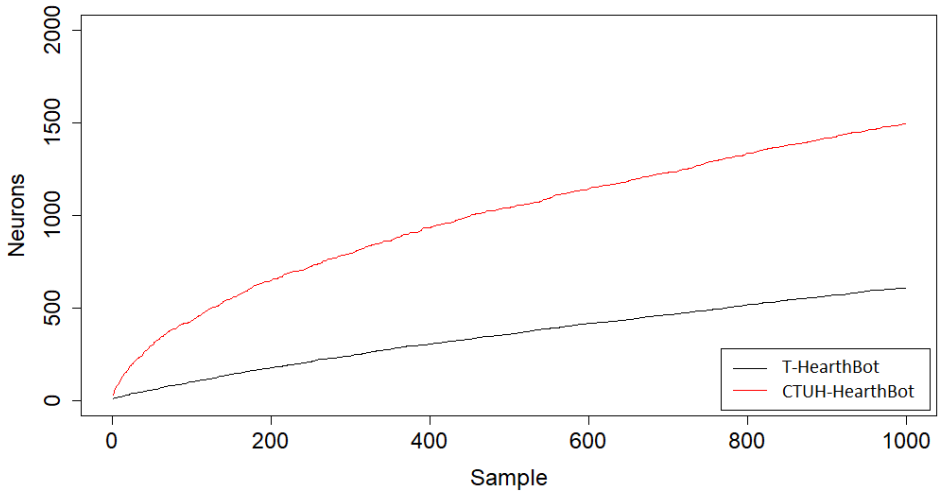
In order to analyze if the creative HARP system is able to better explore an environment search space, an experiment was conducted in a controlled environment on Metastone, for the Shaman vs MCTS Paladin match, and it is described as follows. This experiment measures how many actions are explored and learned by an HARP system, CTH-HearthBot and an ART system, T-HearthBot. By measuring the learned actions it will be possible to know how often the system is exploring new ones. However, in order to do that without randomness interference, the resonance for the action and environment fields was configured as 100% and for the reward field to 0%, thus ensuring that the HARP will learn new environments. In addition, to exclude any possibility of generating a visible cognitive code growth through the performing mode, after ϵ reaching 0, the ϵ decay for this experiment was configured as 0. With this configuration, and a learning rate equal to 1, the HARP system will learn all states without sharing neurons and without being influenced by randomness provided by the performing mode.

As depicted in Chart 20, the behavioral ART system displayed a raw cognitive code growth in a quasi linear slope leading to a maximum near 500 neurons. On the other hand, the Harp displayed a logarithmic growth leading to a maximum near 1500 neurons. These results, as aforementioned, represent the raw cognitive code growth without randomness interference, thus it shows how often the ART system and the HARP system are exploring new actions. The behavior of the HARP system displayed a maximum difference near 50%, if comparing to the ART system, for a maximum of 2000 neurons. Conclusively, for the studied match, the HARP system can in fact explore more actions during its exploration phase.

For the spectrum model, as illustrated in Chart 21, the Shaman versus MCTS Paladin displayed a logarithmic growth as the behavioral model. However, the slope of the aforementioned growth was higher for the spectrum model, and it explores more actions than the behavioral one. The ART based system, T-HearthBot, displayed a maximum neuron count of 1450, while the maximum observed neuron count for the HARP system was 2000. There was observed a growth difference of 27% between both systems, considering a maximum of 2000 neurons. This behavior was expected, since the controlled testing environment enables to measure the raw growth without external disturbances.

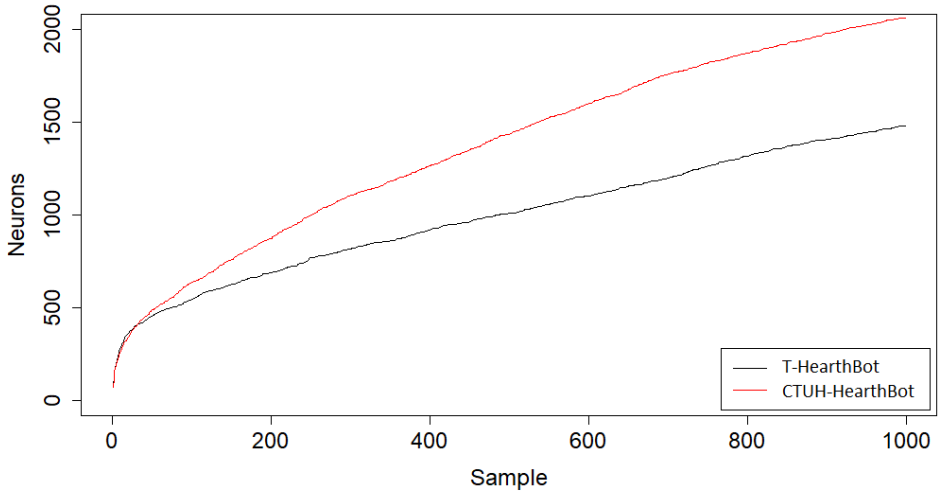
The obtained results indicate that there are, in fact, evidence that the behavior of

Chart 20: Shaman vs Paladin cognitive code growth divergence, between a behavioral HARP and an ART system, when exploring the search space.



Source: By the Author.

Chart 21: Shaman vs Paladin cognitive code growth divergence, between a spectrum HARP and an ART system, when exploring the search space.



Source: By the Author.

the HARP system can improve the ability of an agent that will permit it in better explore its search space on the simulated POMDP, which is a sign of novelty, from the creative system, acting as a tool that helps to reach novel States. However, this can be assumed for the Hearthstone environment used to conduct the aforementioned experiments, since no other investigation was conducted.

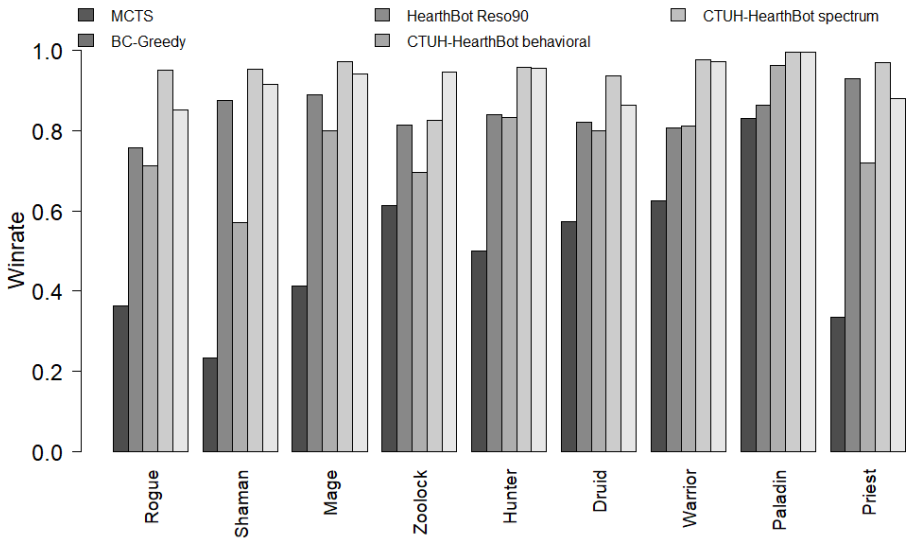
11.4.7 General win rate analysis for CTUH-HearthBot

The general performance for the temporal creative HearthBot was evaluated with CTUH-HearthBot, because it scored higher as shown by the win rate convergence tests presented in Sections 11.4.4, 11.4.5 and in Appendix A, that represents the proposed HARP and UAM and incorporates the temporal Q-Learning FALCON model described in this research. The conducted experiment, in order to verify CTUH-HearthBot behavior, was the same as the one conducted for the training experiment to evaluate HearthBot, where each hero, from Table 2, plays against all others from the same table. Each win rate average was obtained after performing 9000 matches, where each 1000 represents an epoch against one enemy. The CTUH-HeartBot was tested in its two forms, behavioral and spectrum in order to evaluate the general average performance of a model that learns with preconceived behaviors and one that learns from scratch, where both bots were evaluated against the MCTS and BC-Greedy.

As depicted in Chart 22, the CTUH-HearthBot that used the behavioral model displayed an average performance above 90% for all decks, excluding the Zoolock that received an average near 76%. On the other hand, the spectrum model received an average win rate for all decks above 80%. In special, the Mage, Hunter, Warrior and Paladin scored above 90%.

The displayed behavior for both, behavioral and spectrum showed in Chart 22, were higher than the ones observed for HearthBot. Furthermore, CTUH-HearthBot combines the proximity method on its categorization mechanism in its action channel 1, to better categorization, with the HARP system search space exploration behavior, to better explore the search space, thus it was expected that it went better on this test. Considering the displayed behavior of those systems on the analyzed matches in this chapter, was also expected lower performance of the MCTS against CTUH-HearthBot, since it did not performed well for all analyzed matches.

Chart 22: Win rate for CTUH-HearthBot playing with all heroes against the Monte Carlo Tree Search.

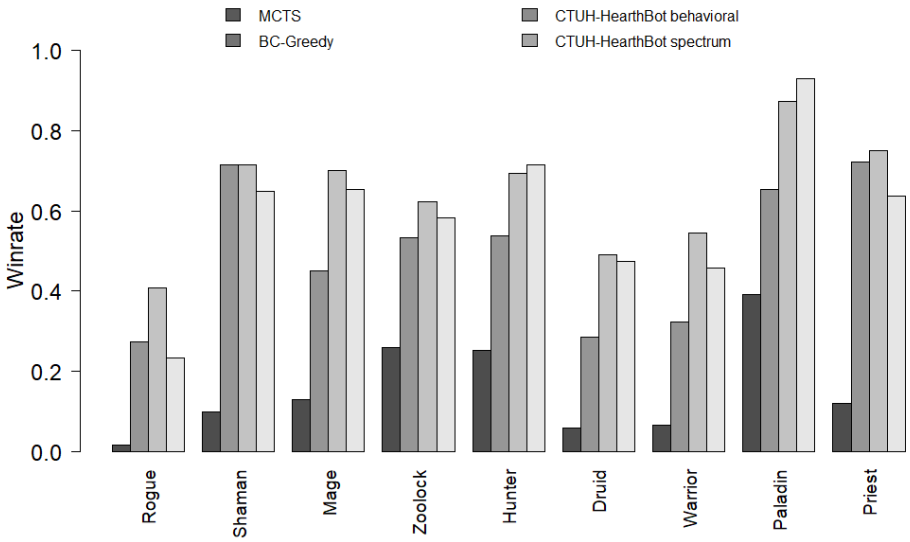


Source: By the Author.

On the other hand, as depicted in Chart 23, the behavioral CTUH-HearthBot average win rate playing against the BC-Greedy was bellow 60% for the Shaman, Mange, Hunter, Paladin and Priest, for all other decks its average win rate was observed above 50% excluding for the Zoolock that obtained an average near 55%. The spectrum model scored above 60% for the same decks, where it obtained 10% more win rate in average for the Zoolock. In contrast, it performed almost 20% worst than the behavioral for the Rogue, Druid and Priest decks. Despite learning from zero, it was not enabled to overcome the behavioral for each match. That stems from the fact that the spectrum model relies in more simulations for learning than the behavioral one, since it learns from scratch. Furthermore, it may be difficult for it to learn complex strategies considering the large amount of states and paths for Hearthstone.

It is important to note that HearthBot, as presented in Section 11.2, obtained an average win rate bellow 1% for all decks, thus it is not suitable to play against the BC-Greedy. When playing against the MCTS the average performance was higher than the ones obtained for the MCTS and slightly higher than the ones obtained for the BC-Greedy. Nevertheless, it seems that CTUH-HearthBot displayed an average win rate convergence similar or better to the one obtained for the BC-Greedy, when playing against the BC-Greedy, and a better average performance if compared with MCTS. There is much to enhance on the proposed CTUH-HearthBot and its variants, which respect on how they handle a reasoning process. However, it was able to play against two Metastone agents

Chart 23: Win rate for CTUH-HearthBot playing with all heroes against the BC-Greedy.



Source: By the Author.

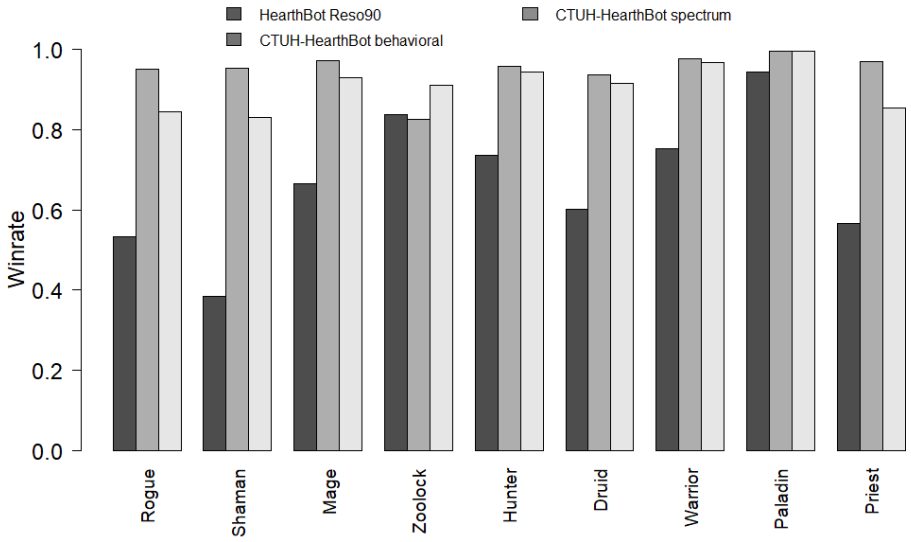
and displayed an average win rate performance above 60%, against the BC-Greedy, and 90%, against the Metastone MCTS, where a substantial win rate gain was observed if compared to HearthBot, MCTS and BC-Greedy versus MCTS cases.

11.4.8 Overall performance against unobserved decks

As illustrated in Chart 24, each bar on the X-Axis is the average win rate, obtained through the selected simulation quantity described in Section 11.4.2, for a hero versus all others. When playing against unobserved decks, CTUH-HearthBot displayed an average performance bellow 80%. The Zoolock deck for the behavioral CTUH-HearthBot, when playing against the unobserved ones, received the lowest average win rate of 82%. By contrast, the Paladin deck received 99% for the behavioral and spectrum models. When analyzing the performance of the behavioral model it can be concluded it is able to enhance an agent's performance over time, but the fact that it relies on handcrafted behaviors need to be taken into consideration, thus its performance is bounded to the quality of the deployed behaviors. However, when analyzing the spectrum model it can be concluded the CTUH-HearthBot is able to generate strategies to play against the MCTS. Finally, the fact that the performance of the MCTS, when performing in Hearthstone, is poor if comparing it to board control strategies, as the one used by BC-Greedy, need to be taken into consideration. Nevertheless, the average performance of CTUH-HearthBot for unobserved decks was better than the one observed for HearthBot that in its worst case

received an average win rate above 40%.

Chart 24: CTUH-HearthBot versus MCTS win rate when performing against unknown decks.

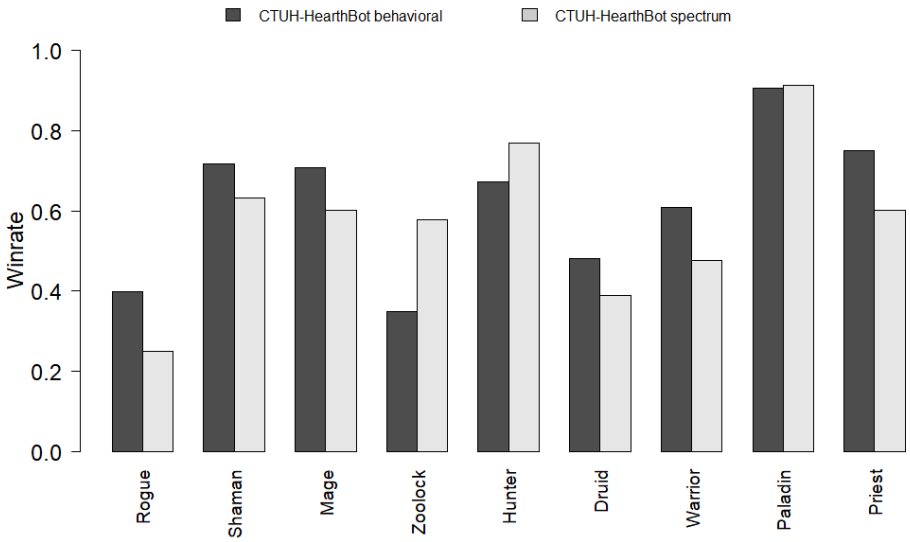


Source: By the Author.

The results depicted in Chart 25, represent the same kind of average as the one presented for Chart 24, however, it represents the average when playing against the BC-Greedy with unobserved decks. The performance of CTUH-HearthBot for all bots was above 38%, where for the worst case it was 38% and for the best case it was 92%. In special, the Shaman, Mage, Zoolock, Hunter, Paladin and Priest went above 60% average win rate while the Rogue, Druid and Warrior went bellow it. The Behavioral model went better when performing with the Rogue, Shaman, Mange, Druid, Warrior and Priest decks. It can be noted that when playing with the Zoolock deck, the spectrum model obtained a 21% better win rate performance than the behavioral. This behavior can be explained as the ability for the spectrum to learn specific strategies, that the behavioral can not encompass, since it was conceived with handcrafted behaviors.

When playing against the MCTS unknown enemies, CTUH-HearthBot performed better than HearthBot. However, the results against the BC-Greedy were oscillating according to the deck strategy. Nevertheless, HearthBot is unable to play against the BC-Greedy, thus there were a substantial gain for it temporal counterparts, since either Metastone handcrafted agents can not went well when playing against it.

Chart 25: CTUH-HearthBot versus Board Control Greedy win rate tendency when performing against unknown decks.



Source: By the Author.

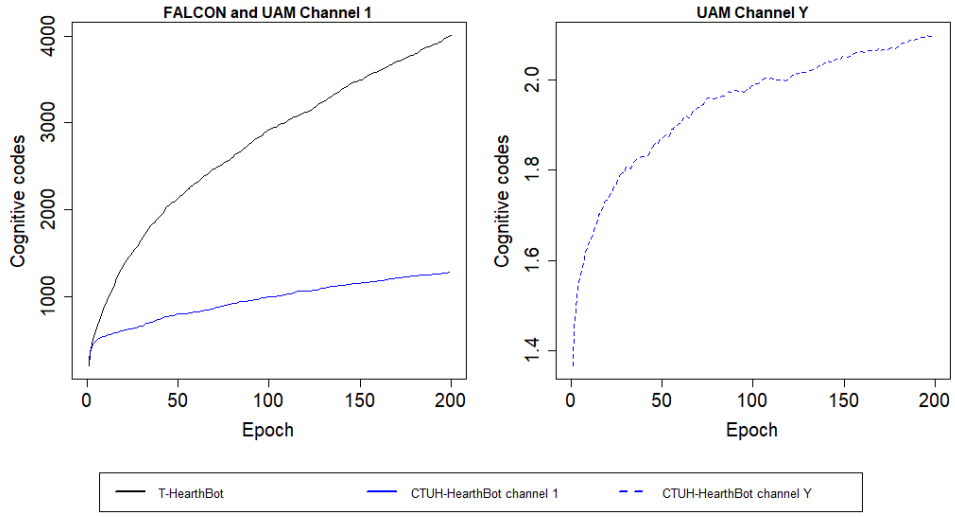
11.4.9 Cognitive code analysis

In order to verify if the UAM-ANN permits the HARP to spend less storage space when performing, the cognitive code growth for CTUH-HearthBot was measured. The conducted experiment was realized for the spectrum model and compared to the T-HearthBot since it represents the FALCON architecture deployed with an ARAM. The verified match was the Hunter versus MCTS Paladin that was selected a priori. The growth behavior for the CTUH-HearthBot was obtained from action channel 1 and in average for environment channel Y.

As showed in Chart 26, T-HearthBot displayed the highest growth than CTUH-HearthBot. The T-HearthBot obtained a maximum observed amount of 4000 cognitive codes, while CTUH-HearthBot obtained a maximum count near 1000 cognitive codes for its action channel 1. The CTUH-HearthBot displayed an average cognitive code count of 2 for its environment channel Y, what indicates that it is sharing information between fields from channel 1 and Y in order to represent fully the HearthStone context.

The growth behavior of the CTUH-HearthBot, representing the UAM-ANN, was the lowest observed one for the collected data. These result show that the T-HearthBot, deployed with the proposed UAM-ANN, can obtain the same of better win rate performance results than the T-HearthBot, based purely on the FALCON and ARAM models, and it also spend less space. Considering that the creative CTUH-HearthBot, deployed with

Chart 26: Hunter vs Paladin cognitive code growth divergence.



Source: By the Author.

the HARP, explores better HearthStone's search space and consequently demanding more storage space, as discussed in Section 11.4.6, the proposed UAM is able to represent that extra demand more efficiently. Finally, as a consequence of its representativeness with a compact structure, it is expected that the CTUH-HearthBot performs faster, as presented in Appendix B, in terms of execution time than the T-HearthBot.

12 CONCLUSIONS

This chapter is divided into three sections, where a discussion over the proposals drawbacks and advantages are provided, described as follows.

a) In Section 12.1, an overview of the conducted research is provided, where it presents the main contributions of the conducted research in a summarized way.

b) In Section 12.2, general discussions over all the proposed HearthBots are presented in this section.

c) Finally, in Section 12.3 general applications are suggested. In addition it is also discussed how the system could be integrated with computer vision and deep learning approaches.

12.1 Overview

In this thesis is proposed a computational model of The Honing Theory of creativity, that has never been attempted before, and deploy it in a system we called HoningStone. This model is guided by search process and a metric based on the Bayesian surprise and a context-specific usefulness metric. This network was applied to the digital collectible card game Hearthstone, which is one of the most popular of its genre. The challenge was to generate creative card combos, a set of cards that when played together outpasses their individual powers. This implies in searching a vast space of possible card combos, which makes traditional approaches inefficient. Moreover, it is also proposed an emergent solution, not only to build combos but to generate broad strategies which at first are just efficient, not creative. In this new scenario, the search space is even larger due to the number of cards and actions combinations and also due to randomness. So we proposed a collection of ANNs, fuzzy ARAM, and ARTMAP, that were deployed in an agent we called Hearthbot, an autonomous agent that plays Hearthstone. Finally, it is proposed a neural reasoning system, called HARP, able to make Hearthbot also creative and more efficient in terms of win rate performance. It relies on incorporating The Honing Theory within the organic learning process of the proposed ANNs, where the associative mode was implemented as neuron activations and the analytic mode, neuron inhibitions. Moreover, the agent actions were also not purely random selected when exploring new possibilities, instead, it used the Bayesian surprise to unveil possibilities not previously attempted.

12.2 General discussion

Although computational intelligence has been widely applied to digital games, in digital collectible card games there is plenty of room to be explored. In addition to it, computational creativity in games is in its infancy. HoningStone, a creative system that applies the Honing theory of creativity to generate creative card combos for *Hearthstone* was able to generate combos that are more creative than a greedy randomized algorithm. As a final remark, although the combos generated by HoningStone are certainly new and playable from an expert point of view, their efficiency has yet to be proven in real games.

HearthBot performance, as evaluated in Section 11.3 from Chapter 11, shows that it was able to overcome Metastone MCTS by at least a 20% win rate margin. However, its performance was unable to defeat the BC-Greedy, that is a simplistic method used to take decisions. That stems from the fact that HearthBot is just a memory, that stores action Q-Values, and do not code complex strategies as POMDP paths. In addition, it is important to note that it can encode actions in a precise way with the proximity metric, thus it is able to represent compact actions with the spectrum model. Perhaps, its way in coding actions can be used as a composite operation for other agents, as accomplished for T-HearthBots variants. When analyzing its ability to create and store information about the game, it went well considering that in its most precise configuration, reso-90, it was able to store 1500000 neurons in average and maintain a win rate, for the easiest matches, near 80%.

By contrast, T-HearthBots displayed an average performance superior to the ones observed for HearthBot, MCTS and BC-Greedy from Metastone, for the conducted experiments. This result could have been observed, since T-HearthBots incorporates temporal learning techniques that enables in representing strategy paths in the simulated POMDP. As presented on the results chapter, T-HearthBot produces an average win rate above 90% for the behavioral model and above 60% for the spectrum model. Despite the behavioral model displayed a better performance, in general, the spectrum model seems more suitable when playing complex matches, against the BC-Greedy, since it can learn strategies from scratch. If considering that the behavioral model can generalize any behavior to be performing in any observed environment, its performance will always be reasonable. That stems from the fact that, when performing with behaviors, any selected one will lead into good POMDP states. Conclusively, any learning model based on

pre-conceived behaviors can not me trustful to verify the quality of the generated solution.

The observed results for, TU-HearthBot, CTH-HearthBot and CTUH-HearthBot could be explained as the ability it in represent actions precisely, since the proximity metric can increase the average win rate obtained when performing. Despite being able to increase the win rate for the performed experiments, when analyzing the behavior of all neuron activation functions between simulation epochs, the ART I displayed similar performance than the proximity metric. This result indicates that the proximity metric, for the conducted experiment match described in results chapter, can produce slightly better win rates since its deviation is higher than the ones obtained for the other metrics. However, using the ART I operation is still useful to categorize environment and achieve generalization. Nevertheless, it seems that the ART II operation can not deal well with the spectrum model and can perform better for the behavioral one for the conducted experiments.

In special, the creative CTH-HearthBot, one of the creative HearthBots, displayed a better average performance for the win rate convergence tests against the BC-Greedy than the T-HearthBot. The action categorization mechanism deployed for this bot permits in increasing the average performance, however the main intention of a creative system is to generate novel solutions that can affect how its general performance can be optimized. As presented by the conducted experiments from Appendix A and Chapter 11, the proposed HARP system displayed the ability that enables it to better explore the search space, because in a controlled environment, it was able to perform and learn more actions than the ϵ-Greedy heuristic during the same amount of epochs.

By contrast to all other T-HearthBots, TR-HearthBot was able to obtain an average performance near 80% for a behavioral model. However, it was not able to keep its average performance when acting against the BC-Greedy, since it is a more difficult enemy. The behavior of the BC-Greedy, with real precise information about every aspect of the game, prevented the TR-HearthBot in performing well. Another observed aspect of its behavior was the fact that the action mask can be used with a behavioral model, otherwise a large amount of bits will need to be used in order to code the action mask.

In terms of attribute behavior analysis, all T-HearthBots performed in a similar way, where similar attribute count was observed for all of them. Since the total used amount of simulations was based on the presence of a coefficient of variation near 0, it is assumed that the behavior attribute counting graphs are a distribution of attributes and

represents the behavior convergence of the algorithms. Furthermore, some T-HearthBots displayed lower attribute count for the damage dealt and mana spent, and yet obtained the best win rate performance if compared to Metastone agents and also T-HearthBot. This result shows that mana managing and damage are important in order to win a game. In addition, the amount of minions played, hero powers and spells used seems to have affected decks in which those are crucial to maintain its strategy. The main problem of those assumptions is that the information obtained for each attribute count graph does not show the real behavior of the agent, since it can not be said where those attributes are being spent on. However, this kind of analysis seems useful to see the general behavior of the agent when playing Hearthstone, where anomalies can be detected for an eventual debugging in more specific mechanisms of an agent.

Conclusively, the UAM-ANN based T-HearthBots, TU-HearthBot and CTUH-HearthBot, displayed similar performance, as the CTH-HearthBot, since both obtained similar win rate response for all conducted experiments expending less neurons in their action channel. Considering that the proposed UAM-ANN activate areas that it will use to generate predictions, the search space for each activation was reduced drastically, as argued in Chapter 11.

The performance of the creative HearthBot, CTUH-HearthBot was the best observed, not only in terms of execution time, but also by neuron usage, considering that it maintained the average win rate equal or above T-HearthBot. However, the nature of the deck being used to perform in Hearthstone need to be taken into consideration, since if it posses a high quantity of actions the UAM-ANN need to be assembled properly in order to avoid high neuron count for its channel 1 and subsequent ones. It seems that the proximity metric, alongside with the HARP system and the UAM-ANN can possibly improve the agent's performance in general by making it explore better the search space, spend less space to store neurons, perform in less time and produce the same or better win rate results than T-HearthBot. Conclusively, the proposed creative HearthBot was able to subdue its non-creative counterparts, where its architecture can possibly leads to new ways to represent creativity in machines.

12.3 Drawbacks and future work

All proposed systems displayed better performance than the analyzed handcrafted solutions provided by Metastone, where MCTS heuristic represents one of the most used methods for agent reasoning to this date for environments influenced by uncertainties. A future search topic of interest is the creation of a mechanism that is able to generate the UAM-ANN HARP topology in order to optimize better the neuron usage from each of its cluster channels. A future research topic of interest for the aforementioned aspects would be the creation of a T-HearthBot, where the HARP is assembled with one UAM-ANN as discussed in Chapter 10.

With respect to the categorization mechanisms, a future research topic of interest on that field would be the creation of a system, in which resonance is used separately on each variable from a received stimulus. By accomplishing that, is expected that the precise categorization mechanism of the proposed Proximity ANN would be improved. Furthermore, this way for categorization would also have utility as a composite operation that could be used on the UAM-ANN and HARP proposals.

A big improvement that can be accomplished for the proposed Hearthstone models, would be the creation of an action model that shares features. If it was used in one of the proposed HearthBots, it would be expected to decrease even more the neuron usage rate and also increasing the ability of them to generalize under uncertainties. Furthermore, if using a model that shares features, the HARP system can be deployed directly with the Bayesian surprise, that can handle feature sharing well. Conclusively, another future research topic of interest would be the proposal of a feature sharing action coding model.

With respect to the Q-Learning, a future research of interest would be related to find ways to incorporate the same politics of neuron pruning, as presented for the temporal reactive model. By using such a technique it is expected that the system throw away bad neurons, with useless Q-Values, thus enhancing the ability of it in learning new and useful information. In addition, it is also expected from this model that it enhances an agent's performance in terms of average win rate for Hearthstone.

12.3.1 Automatic feature extraction for semantic reasoning

A major research topic of interest is the proposal of a system that is able to represent reality or the game as it is. For example, a computer vision system, as exemplified in

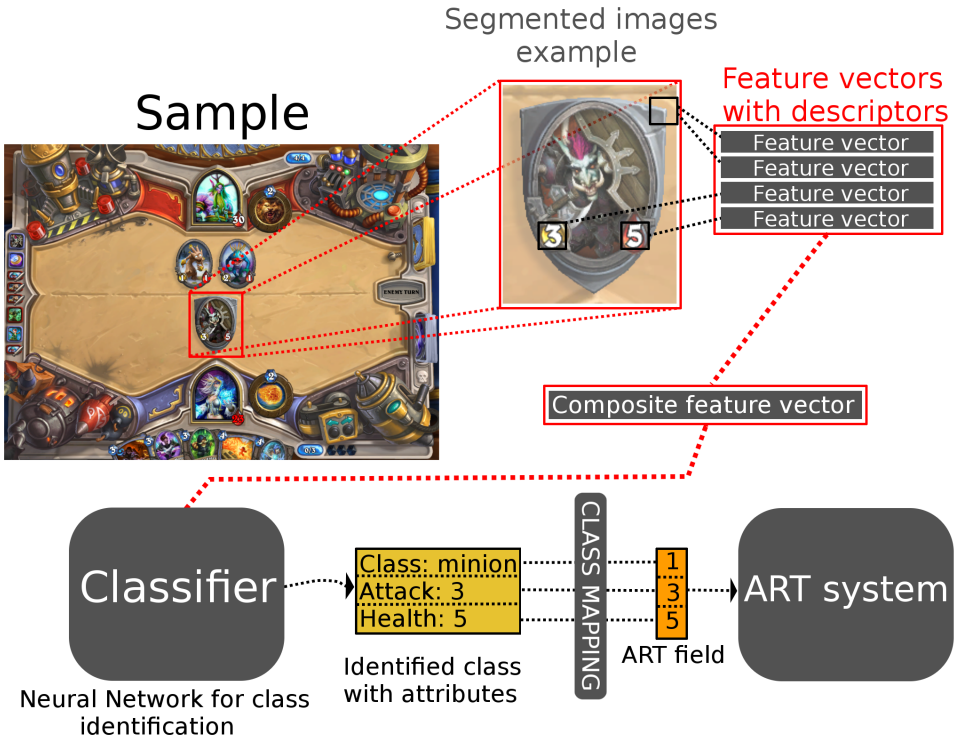
Fig. 12.3.1, if integrated into the proposals from the conducted research the specialist's responsibility of creating preconceived models in a way that an ANN can understand will be reduced. A Convolutional Neural Network could be used to learn image processing filters and thus classifying objects by just looking at them. The obtained semantic information, from the classification process, could be used as input for the proposed UAM-ANN and HARP as discussed on the rest of this section.

In *Hearthstone*, features should be extracted from *Computer Vision* mechanisms in order to represent objects inside the game. The feature extraction mechanism is composed by a complex series of interconnected processes that represents information from what an agent is seeing. In this section, the notion on how features are obtained is given, since it is assumed that the *Hearthstone* feature model for a higher cognition ART system is given by a specialist. As showed in Fig. 12.3.1, the process of a higher cognition ART feature vector extraction starts by receiving the raw sample of a *Time Flow* in the form of an image. This image is filtered in order to enhance its borders, thus facilitating the segmentation of objects into separated small images. As illustrated in Fig. 12.3.1, the segmented image of the minion card in a *Hearthstone* battlefield is filtered again with a filter bank, thus preparing it to extract meaningful information that represents it well.

All meaningful information, from the example depicted in Fig. 12.3.1, is represented by the three black boxes, called interest points, from the segmented image. All the three interest points are used for the extraction of feature vectors, where each one has one or more feature vectors composed by image descriptors. Image descriptors tries to embrace meaningful invariant information from an interest point. Finally, vectors from all interest points are composed in a way that can be used by a classifier. As the final step of feature extraction for a higher *cognition* system, a classifier classifies the segmented image that into a class. For instance, the class that represents the battlefield card on the example showed in Fig. 12.3.1 is a *minion* with 3 *attack* and 5 *health*. This class represents a high level meaning of the raw signals extracted from the image and is assembled in an ART input field with a class mapping function. The mapping function will transform the achieved class and its information into numeric intervals to be used as an ART system's input field.

With this proposal the system can become fully autonomous. Furthermore, it is expected it could learn and performing reasoning through cognition much faster, since the ART based ANNs are faster than Deep Learning Approaches in many ways, as addressed

Figure 32 – Feature extraction example, considering vision, for a minion card on a *Hearthstone* battlefield.



Source: By the Author.

by (TAN, 2004).

12.3.2 General applicability

Other research topics, besides Hearthstone, could be explored in order to verify the behavior of the proposals in a variety of domains. An example of domain that can benefit from the proposed system would be how to solve problems with non-conventional tools. For example, in a hypothetical situation where a space station, with one astronaut inside, caught fire and no fire extinguisher is on board, the proposed HARP system could help the astronaut in generating a solution by novel ways, thus possibly saving his live. It can also control robots to help doctors in finding novel and useful solutions if a unforeseen occur during surgery, the search novel solution can be crucial to avoid a fatality. Furthermore, the proposed system can also be used to generate novel solutions that can be used to prevent planet earth in being destroyed by climate change. A more aggressive utility for the proposed solutions is related to the production of machines that can defeat adversaries during war.

As discussed in this chapter, the proposed solutions from this research can be applied

into a variety of domains. However, ways to incorporate the solutions to cover deficiencies into the public security, health and education would be also an interesting future research topic. The solutions found could be used to help dealing with city management, police forces operations and also to find novel and meaningful ways to deal with government corruption and how to improve the health and education of a population, thus possibly enhancing the general quality of life from an underdeveloped country and even smaller problems from well established ones.

BIBLIOGRAPHY

ABDELLARIF, T. et al. Rigorous design of robot software: A formal componente-based approach. **Robotics and autonomous systems**, 2012. Citado na página 57.

AGUILAR, W.; PéREZ, R. P. y. Early-creative behavior: the first manifestations of creativity in a developmental agent. In: **International Conference on Computational Creativity**. [S.l.: s.n.], 2017. Citado 5 vezes nas páginas 31, 35, 45, 46, and 48.

AMABILE, T. M. The social psychology of creativity: A consensual assessment technique. **Journal of Personality and Social Psychology**, v. 43, n. 5, p. 997–1013, abr. 1982. Citado 2 vezes nas páginas 43 and 46.

AMORIM, A. et al. Creative flavor pairing: Using rdc metric to generate and assess ingredients combinations. In: **International Conference on Computational Creativity**. [S.l.: s.n.], 2017. Citado 7 vezes nas páginas 31, 35, 39, 45, 46, 47, and 48.

AUGELLO, A. et al. Creative robot dance with variational encoder. In: **International Conference on Computational Creativity**. [S.l.: s.n.], 2017. Citado 5 vezes nas páginas 31, 35, 45, 46, and 47.

BAIER, H.; WINANDS, M. H. M. Mcts-minimax hybrids. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 7, n. 2, p. 167–179, June 2015. ISSN 1943-068X. Citado na página 32.

BALDI, P.; ITTI, L. Of bits and wows: A bayesian theory of surprise with applications to attention. **Neural Netw.**, Elsevier Science Ltd., Oxford, UK, UK, v. 23, n. 5, p. 649–666, jun. 2010. ISSN 0893-6080. Disponível em: <http://dx.doi.org/10.1016/j.neunet.2009.12.007>. Citado na página 74.

BASILE, P. et al. Solving a complex language game by using knowledge-based word associations discovery. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 8, n. 1, p. 13–26, March 2016. ISSN 1943-068X. Citado na página 32.

BINATO, S.; OLIVEIRA, G. C.; ARAUJO, J. L. A greedy randomized adaptive search procedure for transmission expansion planning. **IEEE Power Engineering Review**, v. 21, n. 4, p. 70–71, April 2001. ISSN 0272-1724. Citado 2 vezes nas páginas 93 and 241.

BLIZZARD, E. 2018. Available at: https://us.battle.net/hearthstone/en/. Disponível em: <https://us.battle.net/hearthstone/en/>. Citado 2 vezes nas páginas 13 and 123.

BODEN, M. A. **The creative mind myths and mechanisms**. [S.l.]: Routledge, 2004. Citado 2 vezes nas páginas 43 and 73.

BODEN, M. A. Creativity and alife. **Artificial Life**, v. 21, n. 3, p. 354–365, jun. 2015. Citado 2 vezes nas páginas 44 and 46.

BROOKS, R. A. A robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, 1986. Citado na página 57.

BRUNETE, A. et al. A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots. **Robotics and autonomous systems**, 2012. Citado na página 57.

BURSZTEIN, E. **2014**. How to appraise hearthstone card values. Disponível em: <https://www.elie.net/blog/hearthstone/>. Citado 3 vezes nas páginas 48, 49, and 50.

CARPENTER, G. A.; GROSSBERG, S. The art of adaptive pattern recognition by a self-organizing neural network. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 21, n. 3, p. 77–88, mar. 1988. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/2.33>. Citado 2 vezes nas páginas 32 and 67.

CARPENTER, G. A.; GROSSBERG, S.; REYNOLDS, J. Artmap: a self-organizing neural network architecture for fast supervised learning and pattern recognition. In: **Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on**. [S.l.: s.n.], 1991. i, p. 863–868 vol.1. Citado 5 vezes nas páginas 32, 33, 65, 67, and 108.

CARPENTER, G. A.; GROSSBERG, S.; ROSEN, D. B. Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system. **Neural Netw.**, Elsevier Science Ltd., Oxford, UK, UK, v. 4, n. 6, p. 759–771, nov. 1991. ISSN 0893-6080. Disponível em: <http://dx.doi.org/10.1016/0893-6080(91)90056-B>. Citado 3 vezes nas páginas 32, 66, and 67.

CELIKKANAT, H.; ORHAN, G.; KALKAN, S. A probabilistic concept web on a humanoid robot. **Autonomous Mental Development, IEEE Transactions on**, v. 7, n. 2, p. 92–106, Junho 2015. ISSN 1943-0604. Citado na página 32.

CHENG, C.-I.; LIU, D.-M. An intelligent clothes search system based on fashion styles. In: **Machine Learning and Cybernetics, 2008 International Conference on**. [S.l.: s.n.], 2008. v. 3, p. 1592–1597. Citado 4 vezes nas páginas 31, 44, 46, and 47.

COLTON, S. et al. Stakeholder groups in computational creativity research and practice. In: BESOLD, T. R.; SCHORLEMMER, M.; SMAILL, A. (Ed.). **Computational Creativity Research: Towards Creative Machines**. Paris: Atlantis Press, 2015. p. 3–36. Citado 2 vezes nas páginas 44 and 46.

DEMILICH. 2016. Disponível em: <http://www.demilich.net/metastone/>. Citado 2 vezes nas páginas 164 and 172.

DIPAOLA, S. Using a contextual focus model for an automatic creativity algorithm to generate art work. In: SAMSONOVICH, A.; ROBERTSON, P. (Ed.). **5th Annual International Conference on Biologically Inspired Cognitive Architetures**. [S.l.], 2014. (Procedia Computer Science, v. 41), p. 212–219. ISSN 1877-0509. Citado 5 vezes nas páginas 31, 35, 44, 46, and 47.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. **Journal of Global Optimization**, v. 6, n. 2, p. 109–133, 1995. ISSN 1573-2916. Disponível em: <http://dx.doi.org/10.1007/BF01096763>. Citado 2 vezes nas páginas 93 and 241.

FITZGERALD, T.; GOEL, A.; THOMAZ, A. Human-robot co-creativity: Task transfer on a spectrum of similarity. **International Conference on Computational Creativity**, 2017. Citado 4 vezes nas páginas 31, 35, 46, and 48.

GABORA, L. Revenge of the "Neurds": Characterizing Creative Thought in Terms of the Structure and Dynamics of Memory. **Creativity Research Journal**, v. 22, n. 1, p. 1–13, jan. 2010. Citado 4 vezes nas páginas 35, 36, 71, and 73.

GABORA, L. Revenge of the neurds: characterizing creative thought in terms of the structure and dynamics of memory. **Creativity Research Journal**, Taylor & Francis Group, v. 22, n. 1, p. 1–13, 2010. Citado na página 44.

GóES, L. F. W. et al. Honingstone: Building creative combos with honing theory for a digital card game. **IEEE Transactions on Computational Intelligence and AI in Games**, PP, n. 99, p. 1–1, 2016. ISSN 1943-068X. Citado 2 vezes nas páginas 39 and 123.

GOLDSMAN, D.; NANCE, R. E.; WILSON, J. R. A brief history of simulation revisited. In: **Proceedings of the 2010 Winter Simulation Conference**. [S.l.: s.n.], 2010. p. 567–574. ISSN 0891-7736. Citado na página 59.

GRACE, K.; MAHER, M. L. What to expect when you're expecting: The role of unexpectedness in computationally evaluating creativity. In: JOSEF STEFAN INSTITUTE, LJUBLJANA, SLOVENIA. **Proceedings of the Fifth International Conference on Computational Creativity**. Ljubljana, Slovenia: Josef Stefan Institute, Ljubljana, Slovenia, 2014. ISBN 978-961-264-055-2. Disponível em: <http://computationalcreativity.net/iccc2014/wp-content/uploads/2014/06//8.2_Grace.pdf>. Citado na página 73.

GRACE, K. et al. Data-intensive evaluation of design creativity using novelty, value, and surprise. **International Journal of Design Creativity and Innovation**, Informa UK Limited, v. 3, n. 3-4, p. 125–147, ago. 2014. Citado 3 vezes nas páginas 44, 46, and 47.

GROSSBERG, S. **The Adaptive Brain**. [S.l.]: Elsevier, 1989. I and II. Citado na página 33.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowl. Acquis.**, Academic Press Ltd., London, UK, UK, v. 5, n. 2, p. 199–220, jun. 1993. ISSN 1042-8143. Disponível em: <http://dx.doi.org/10.1006/knac.1993.1008>. Citado na página 216.

GULDNER, J.; UTKIN, V. I.; BAUER, R. A three-layered hierarchical path control system for mobile robots: Aalgorithm and experiments. **Robotics and Autonomous Systems**, 1995. Citado na página 57.

HENRIKSEN, J. O. et al. Implementations of time (panel). In: JONES, D. W. (Ed.). **Proceedings of the 18th Conference on Winter Simulation**. New York, NY, USA: ACM, 1986. (WSC '86), p. 409–416. ISBN 0-911801-11-1. Disponível em: <http://doi.acm.org/10.1145/318242.318467>. Citado 2 vezes nas páginas 31 and 58.

JUNIOR, C. R. F. et al. Dependent creativity: A domain independent metric for the assessment of creative artifacts. In: **International Conference on Computational Creativity**. [S.l.: s.n.], 2016. Citado 10 vezes nas páginas 35, 36, 39, 44, 45, 46, 47, 48, 136, and 137.

KIM, H.-S.; CHO, S.-B. Application of interactive genetic algorithm to fashion design. **Engineering Applications of Artificial Intelligence**, v. 13, n. 6, p. 635 – 644, 2000. ISSN 0952-1976. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0952197600000452>. Citado 5 vezes nas páginas 31, 35, 44, 46, and 47.

KOWALIW, T.; DORIN, A.; MCCORMACK, J. Promoting creative design in interactive evolutionary computation. **IEEE Transactions on Evolutionary Computation**,

v. 16, n. 4, p. 523–536, AUG 2012. ISSN 1089-778X. Citado 6 vezes nas páginas 31, 35, 44, 46, 47, and 48.

KUNANUSONT, K.; LUCAS, S. M.; PéREZ-LIéBANA, D. General video game ai: Learning from screen capture. In: **2017 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2017. p. 2078–2085. Citado 5 vezes nas páginas 33, 45, 51, 52, and 54.

MACEDO, L.; CARDOSO, A. Modeling forms of surprise in an artificial agent. In: MOORE, J. D.; STENNING, K. (Ed.). **Proceedings of the 23rd Annual Conference of the Cognitive Science Society**. Edinburgh: Lawrence Erlbaum Associates, 2001. p. 588–593. Citado 3 vezes nas páginas 43, 46, and 47.

MACEDO, L.; REISENZEIN, R.; CARDOSO, A. Modeling forms of surprise in artificial agents: empirical and theoretical study of surprise functions. In: FORBUS DEDRE GENTNER, T. R. K. (Ed.). **Proceedings of the 26th Annual Conference of the Cognitive Science Society**. Mahwah: Lawrence Erlbaum, 2004. p. 873–878. Citado 3 vezes nas páginas 43, 46, and 47.

MATUSZEK, C. et al. An introduction to the syntax and content of cyc. In: **Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering**. [S.l.: s.n.], 2006. p. 44–49. Citado na página 32.

MINSKY, M. **A Framework for Representing Knowledge**. Cambridge, MA, USA, 1974. Citado na página 215.

MIYASHITA, S. et al. Developing game ai agent behaving like human by mixing reinforcement learning and supervised learning. In: **2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)**. [S.l.: s.n.], 2017. p. 489–494. Citado 7 vezes nas páginas 31, 32, 33, 45, 51, 52, and 54.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 00280836. Disponível em: <http://dx.doi.org/10.1038/nature14236>. Citado 3 vezes nas páginas 31, 50, and 54.

ORTONY, A.; PARTRIDGE, D. Surprisingness and expectation failure: what's the difference? In: KAMBHAMPATI, S. (Ed.). **Proceedings of the Tenth International Joint Conference on Artificial Intelligence**. São Paulo: AAAI Press, 1987. p. 106–108. Citado 2 vezes nas páginas 43 and 46.

PARTRIDGE, D. Input-expectation discrepancy reduction: A ubiquitous mechanism. In: **Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1**. San Francisco: Morgan Kaufmann Publishers Inc., 1985. (IJCAI'85), p. 267–273. Citado 2 vezes nas páginas 43 and 46.

POUNDSTONE, W. **The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge**. [S.l.]: Courier Corporation, 2013. Citado na página 125.

RAMOS, S. L.; GOéS, L. F. W. Avaliação da percepção de jogadores sobre a criatividade de combos do jogo digital de cartas hearthstone. In: **Brazilian Computing Society conference on Games (SBGames)**. [S.l.: s.n.], 2016. Citado na página 39.

RITCHIE, G. Assessing creativity. In: **AISB Symposium on artificial intelligence and creativity in arts and science**. York, England: [s.n.], 2001. p. 3–11. Citado 2 vezes nas páginas 34 and 35.

SAFFIOTTI, A. et al. The peis-ecology project: Vision and results. In: **Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on**. [S.l.: s.n.], 2008. p. 2329–2335. Citado na página 32.

SCHMIDHUBER, J. Formal theory of creativity, fun, and intrinsic motivation (1990-2013;2010. **Autonomous Mental Development, IEEE Transactions on**, v. 2, n. 3, p. 230–247, Sept 2010. ISSN 1943-0604. Citado na página 35.

SEPHTON, N. et al. Heuristic move pruning in monte carlo tree search for the strategic card game lords of war. In: **2014 IEEE Conference on Computational Intelligence and Games**. [S.l.: s.n.], 2014. p. 1–7. ISSN 2325-4270. Citado 2 vezes nas páginas 48 and 49.

SILVA, A. R.; GóES, L. F. W. Hearthbot: An autonomous agent based on fuzzy art adaptive neural networks for the digital collectible card game hearthstone. **IEEE Transactions on Computational Intelligence and AI in Games**, PP, n. 99, p. 1–1, 2017. ISSN 1943-068X. Citado na página 39.

SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. **Nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, jan. 2016. Citado 7 vezes nas páginas 31, 32, 33, 49, 50, 54, and 125.

SONDIK, E. J. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. **Oper. Res.**, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 26, n. 2, p. 282–304, abr. 1978. ISSN 0030-364X. Disponível em: <http://dx.doi.org/10.1287/opre.26.2.282>. Citado na página 61.

SOWA, J. F. Principles of semantic networks. In: . [S.l.]: Morgan Kaufmann, 1991. Citado na página 216.

STEEG, M. V. D.; DRUGAN, M. M.; WIERING, M. Temporal difference learning for the game tic-tac-toe 3d: Applying structure to neural networks. In: **Computational Intelligence, 2015 IEEE Symposium Series on**. [S.l.: s.n.], 2015. p. 564–570. Citado na página 32.

STIENSMEIER-PELSTER, J.; MARTINI, A.; REISENZEIN, R. The role of surprise in the attribution process. **Cognition & Emotion**, Taylor & Francis, v. 9, n. 1, p. 5–31, mar. 1995. Citado 3 vezes nas páginas 43, 46, and 47.

SUGIMOTO, N. et al. Trax solver on zynq with deep q-network. In: **Field Programmable Technology (FPT), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 272–275. Citado na página 32.

SZYGENDA, S. A.; HEMMING, C. W.; HEMPHILL, J. M. Time flow mechanisms for use in digital logic simulation. In: **Proceedings of the 5th Conference on Winter Simulation**. New York, NY, USA: ACM, 1971. (WSC '71), p. 488–495. Disponível em: <http://doi.acm.org/10.1145/800294.811476>. Citado na página 57.

TAN, A. H. Adaptive resonance associative map: a hierarchical art system for fast stable associative learning. In: **Neural Networks, 1992. IJCNN., International Joint Conference on**. [S.l.: s.n.], 1992. v. 1, p. 860–865 vol.1. Citado 3 vezes nas páginas 66, 67, and 68.

TAN, A.-H. Adaptive resonance associative map. **Neural Networks**, v. 8, n. 3, p. 437 – 446, 1995. ISSN 0893-6080. Citado 5 vezes nas páginas 32, 65, 66, 67, and 108.

TAN, A.-H. Falcon: a fusion architecture for learning, cognition, and navigation. In: **Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on**. [S.l.: s.n.], 2004. v. 4, p. 3297–3302 vol.4. ISSN 1098-7576. Citado 17 vezes nas páginas 32, 33, 45, 52, 53, 54, 55, 57, 65, 66, 67, 68, 141, 143, 145, 177, and 199.

TAN, A. hwee et al. Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. **IEEE Transactions on Neural Networks**, p. 230244, 2008. Citado 9 vezes nas páginas 50, 52, 53, 54, 55, 57, 65, 68, and 172.

TENG, T. H.; TAN, A. H. Fast reinforcement learning under uncertainties with self-organizing neural networks. In: **2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)**. [S.l.: s.n.], 2015. v. 2, p. 51–58. Citado 6 vezes nas páginas 31, 33, 45, 53, 54, and 55.

TENORTH, M.; BEETZ, M. Knowrob: knowledge processing for autonomous personal robots. In: **Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on**. [S.l.: s.n.], 2009. p. 4261–4266. Citado na página 32.

TEYTAUD, O.; FLORY, S. Upper confidence trees with short term partial information. In: _____. **Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, April 27-29, 2011, Proceedings, Part I**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 153–162. ISBN 978-3-642-20525-5. Citado na página 48.

TURING, A. M. Computers & thought. In: FEIGENBAUM, E. A.; FELDMAN, J. (Ed.). Cambridge, MA, USA: MIT Press, 1995. cap. Computing Machinery and Intelligence, p. 11–35. ISBN 0-262-56092-5. Disponível em: <http://dl.acm.org/citation.cfm?id=216408.216410>. Citado na página 7.

VARSHNEY, L. et al. Cognition as a part of computational creativity. In: **International Conference on Cognitive Informatics and Cognitive Computing**. [S.l.: s.n.], 2013. p. 36–43. Citado 6 vezes nas páginas 31, 35, 44, 46, 47, and 48.

VELDE, F. van der et al. A semantic map for evaluating creativity. In: TOIVONEN SIMON COLTON, M. C. D. V. H. (Ed.). **Proceedings of the Sixth International Conference on Computational Creativity**. Provo: Brigham Young University, 2015. p. 94–101. Citado 2 vezes nas páginas 44 and 46.

WANG, D.; TAN, A. H. Creating autonomous adaptive agents in a real-time first-person shooter computer game. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 7, n. 2, p. 123–138, June 2015. ISSN 1943-068X. Citado 13 vezes nas páginas 32, 57, 65, 67, 68, 78, 81, 85, 108, 145, 146, 147, and 148.

WANG, W.; TAN, A. H.; TEOW, L. N. Semantic memory modeling and memory interaction in learning agents. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, PP, n. 99, p. 1–14, 2017. ISSN 2168-2216. Citado na página 55.

WARD, C. D.; COWLING, P. I. Monte carlo search applied to card selection in magic: The gathering. In: **2009 IEEE Symposium on Computational Intelligence and Games**. [S.l.: s.n.], 2009. p. 9–16. ISSN 2325-4270. Citado na página 32.

WARD, C. D.; COWLING, P. I. Monte carlo search applied to card selection in magic: The gathering. In: **Proceedings of the 5th International Conference on Computational Intelligence and Games**. Piscataway, NJ, USA: IEEE Press, 2009. (CIG'09), p. 9–16. ISBN 978-1-4244-4814-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1719293.1719306>. Citado 2 vezes nas páginas 48 and 49.

WARD, C. D.; COWLING, P. I. Monte Carlo search applied to card selection in Magic: The Gathering. In: **Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on**. IEEE, 2009. p. 9–16. ISBN 978-1-4244-4814-2. Disponível em: <http://dx.doi.org/10.1109/cig.2009.5286501>. Citado 2 vezes nas páginas 48 and 49.

WENG, J. Symbolic models and emergent models: A review. **IEEE Transactions on Autonomous Mental Development**, v. 4, n. 1, p. 29–53, March 2012. ISSN 1943-0604. Citado na página 32.

WENG, J. Symbolic models and emergent models: A review. **Autonomous Mental Development, IEEE Transactions on**, v. 4, n. 1, p. 29–53, March 2012. ISSN 1943-0604. Citado 3 vezes nas páginas 44, 47, and 57.

WENG, J.; LUCIW, M.; ZHANG, Q. Brain-like emergent temporal processing: Emergent open states. **Autonomous Mental Development, IEEE Transactions on**, v. 5, n. 2, p. 89–116, Junho 2013. ISSN 1943-0604. Citado 2 vezes nas páginas 32 and 57.

YANNAKAKIS, G. N.; TOGELIUS, J. A panorama of artificial and computational intelligence in games. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 7, n. 4, p. 317–335, Dec 2015. ISSN 1943-068X. Citado na página 32.

ZHANG, Y.; YANG, H. An application of radial basis function network and genetic algorithm to fashion design system. In: **Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on**. [S.l.: s.n.], 2013. p. 84–88. Citado 5 vezes nas páginas 31, 35, 44, 46, and 47.

ZHAO, D. et al. Deep reinforcement learning with experience replay based on sarsa. In: **2016 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.: s.n.], 2016. p. 1–6. Citado 5 vezes nas páginas 32, 33, 45, 52, and 54.

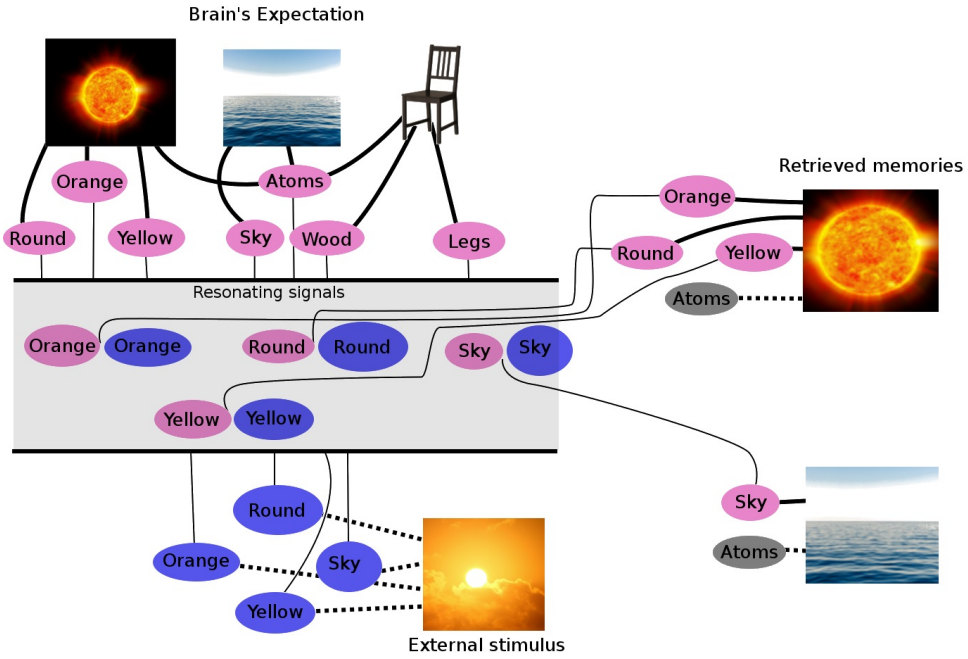
Appendix

APPENDIX A – THE ADAPTIVE RESONANCE THEORY PRACTICAL EXAMPLE

In this appendix is presented a practical interpretation of the ART in form of examples. Furthermore, it is also discussed how its aspects can be deployed using data structures in a reasoning process.

The ART process, illustrated in Figure 33, starts after receiving an external stimulus image of a sunset from one or more sensors. The received image is analyzed and broken into understandable symbols called: *Round* representing the sun's curvature; *Sky* representing the clouds; *Yellow* representing the sky and sun's colors; *Orange* also representing the sky and sun's colors. Those symbols triggers a brain mechanism called *Coherence Phase Locking*, where all neurons fire simultaneously.

Figure 33 – Adaptive Resonance Theory scheme.



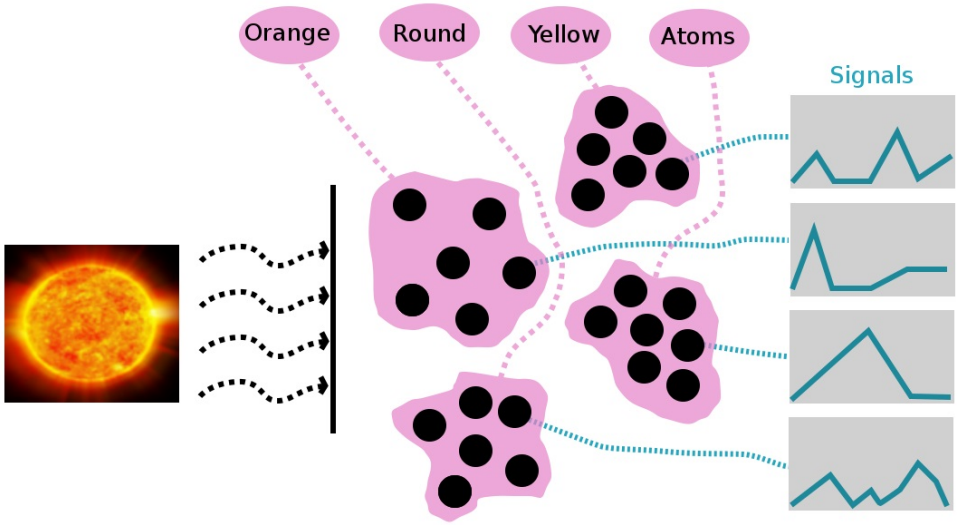
Source: By the Author.

When in *Coherence Phase Locking*, all neurons activate fragmented memories already stored in it and represented by the symbols: *Round*; *Orange*; *Yellow*; *Atoms*; *Sky*; *Wood*; *Legs*. All those triggered fragmented memories that were fired interacts with the external stimulus and this process is called *Resonance Checking*. When the *Resonance Checking* is happening, symbols are grouped by similarity and those that does not group with others are inhibited by an inhibition process. If a symbol is grouped with others,

then they are called resonating symbols. If a resonating symbol is already stored inside the brain, then it will activate a full memory as can be seen on the right side of Figure 33, where the memories about the sun and the ocean were retrieved. Memories can be retrieved by receiving incomplete information about an environment. After retrieving memories, the ART also stores or reinforces the received external stimulus by learning, assembling new memories.

All symbols, as depicted in Figure 33, are defined in this research as information formed by a complex network of interconnected neuron clusters, sets of meaningful informations, assembled with any kind of pattern recognition neural network, or even raw data obtained from an agent's sensors. For instance, the example showed in Figure A shows that the image of the sun can be represented by the symbols *Orange*, *Round*, *Yellow*, and *Atoms*, where each one of them was obtained from neuron clusters. The response or represented stimulus, in a neuron cluster, is coded as a signal that can be used in an ART system. This way of coding symbols facilitates to create mechanisms for pattern recognition, thus to perform a *Resonance Checking* in a digital computer.

Figure 34 – Adaptive Resonance Theory symbols as signals represented by a complex network of interconnected neurons.

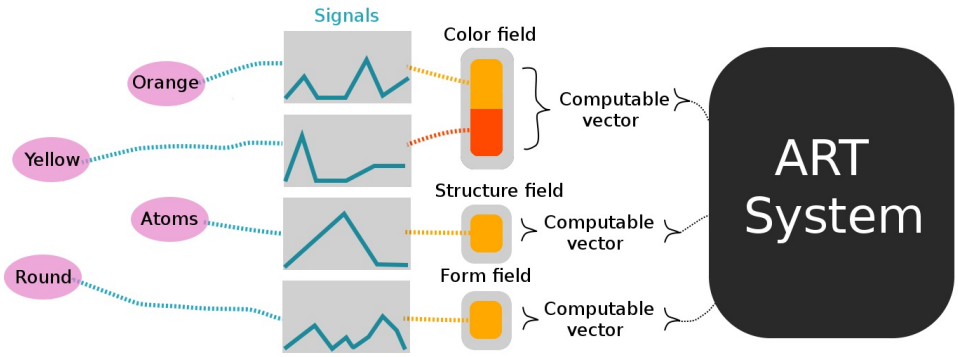


Source: By the Author.

There are two essential problems when dealing with an ART system to simulate a POMDP to develop strategies to control an agent. Firstly, there is a lack of a standard structure to store a template of which symbols the system should check resonance with, in which order and class, secondly, the resonating mechanism retrieves information, but

it does not, represent semantic information in its own structure, thus not representing adequately a State. To tackle the external stimulus storage problem, any ART established system organizes its symbols inside *Feature Fields*, where each field is composed of feature vectors that represent input signals. For example, Figure 35 shows that the symbols *Orange* and *Yellow* represent colors, thus if the system needs to retrieve memories that are related to the color present on an environment, then the color of each pixel should be stored in a field. This mechanism is used inside the *Resonance Checking* procedure and it recognizes what expectations should be paired inside the process without a information handling mechanism.

Figure 35 – Adaptive Resonance Theory signals organized inside representative fields.



Source: By the Author.

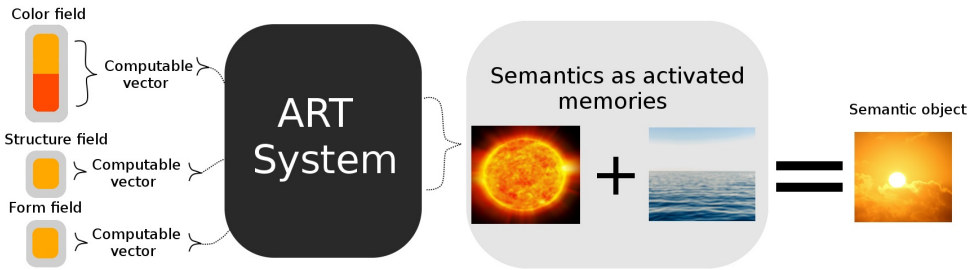
In order simulate a POMDP, *fields* are used to organize information about each POMDP component, thus representing states, rewards and actions. On the contrary, transitions are unrepresented explicitly by any field, because the system needs to be dynamic and adaptable in a way it does stay unbounded to a predefined *Time Flow*. This way of arranging *fields* is in conformity with a FALCON that works on top of an ART system.

A.1 Semantics as retrieved memories

In a POMDP, semantics can be seen as a path obtained from the reasoning process since it represents a sequence of *actions* that an agent should take in order to trigger a coherent sequence of *events* that will maximize its overall reward over time. From the point of view of an ART system, semantics arises as a consequence of a list of retrieved interrelated memories as depicted in Figure 36, where the received input signal has activated

the sun and ocean memories as interrelated responses from the *Resonance Checking*. This implies on an agent being able to advance semantics not by a POMDP but also in its internal *cognition*.

Figure 36 – Adaptive Resonance Theory semantics as retrieved memories.



Source: By the Author.

In order to organize retrieved memories and represent semantics, in this research, a *Semantic Object* is used to reference a set of symbols. For instance, in Figure 36, the image of the sunset with orange clouds is composed from the semantic objects sun and ocean, where each one of them is composed by *symbols*.

A.2 Representing information

An ART system can have as many fields it requires and to represent whatever is necessary following the application's needs, thus being flexible in representing environments and simulating processes. Each feature vector inside a field can be composed of continuous or binary variables. Discrete values are not typically used, since when deploying an ART system all feature vectors are normalized between [0,1]. Continuous variables are used to represent stimulus obtained from an environment and binary variables are used to the represent presence or absence of a semantic object in an environment.

In general, representing variables are not an issue since they can be directly obtained through sensors. In contrast, if an agent needs to retrieve meaningful semantic information from retrieved memories or translate previously obtained semantic objects into feature vectors to be used as signals, then it seems reasonable to portray them coherently. Organizing information aids in coding a *Semantic Object* into feature vectors. Furthermore, it also aids humans in understanding what that information really portrays in a coherent way, thus facilitating handling it with a computational system.

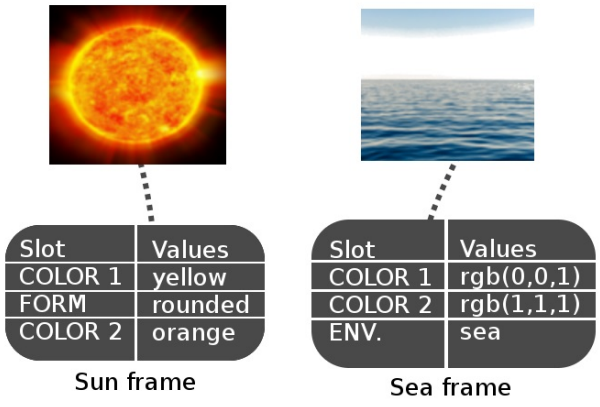
For instance, in Figure 36, the obtained response is composed by two semantic

objects that do need to be combined in order to represent the sunset with clouds as a third *Semantic Object*. In order to aid in representing a *Semantic Object* as a variable, data structures such as *Frames*, *Semantic Networks* and *Ontologies* can be used, thus translating semantic objects into input signals inside feature vectors or in translating activated memories, obtained from the *Resonance Checking*, into semantic objects that can be used to label *actions* that an agent can perform.

A.2.1 Frames

Frames (MINSKY, 1974) are considered in this research as data structures that store information from symbols about an environment and objects in a game. They are used to organize any information about a game inside slots. Each slot is used to dictate a type of variable, as illustrated in Figure 37. The sun is represented by a frame with three slots, where the slots *COLOR1*, *COLOR2*, *FORM* and *ENV.* represent semantic information, of classes. The type of value stored by each slot can vary from string to numbers and are define a priori. For instance, the value for the *COLOR1* slot on the Sun frame have the string value *yellow*, while the *COLOR1* value for the Sea frame is represented by a red-green-blue vector that comprises numbers. Furthermore, each slot can store a reference to other frames, thus representing *Semantic Objects* and sequence of *events*.

Figure 37 – Frame example storing information about the sun and the sea.



Source: By the Author.

A.2.2 Semantic network and Ontologies

Frames can be extracted from *Semantic Networks* (SOWA, 1991), that represents semantics through directed or undirected graphs. Those graphs can be directly extracted from a game's behavior, *event*'s sequences and the relation between objects. A graph is composed of vertices interconnected by edges and defined as $G = (V, E)$, where V is the set of all vertices and E a set of edges. A *Semantic Network* has stereotypes associated with each edge and represents a semantic relation between vertices, thus representing meaning.

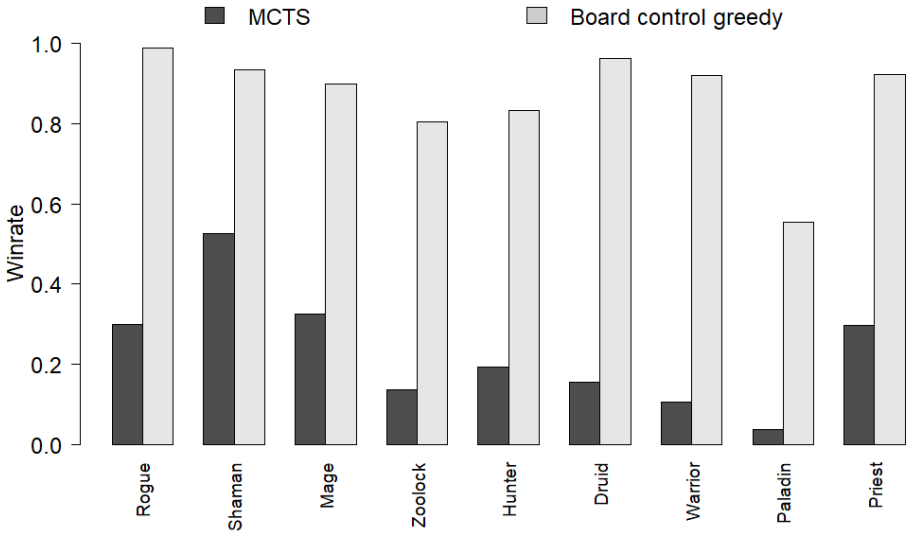
Semantic networks are at a lower level of representation than frames and they do not have a mechanism to naturally maintain references to other *Semantic Networks*, thus it can be confusing in dealing with sequences of *events* that are interrelated with this representation. On the other hand, ontologies (GRUBER, 1993) have no predefined structure and formal definition, they are structures created on demand. In this research, they are used to simplify *Semantic Networks* for specific applications that involve representing semantics that are present on a State obtained from a *Time Flow*.

APPENDIX B – WIN RATE CONVERGENCE

This appendix presents a complement for the convergence test for all T-HearthBots when playing against some of the worst observed cases from HearthBot's evaluation. Some of the worst cases, when playing against Metastone's MCTS were: 1) Aggro Shaman; 2) Tempo Mage; and 3) Face Hunter. Furthermore, in order to test all T-HearthBots against a Board Control Greedy from Metastone, three more decks were selected: 1) Control Warrior; 2) Midrange Druid; and 3) Malygos Rogue. The Board Control Greedy decks were selected based on the worst performance verified when observing how well the MCTS and BC-Greedy performed against it from the experiment conducted in Section 11.2.

As illustrated in Chart 27, the MCTS displays an overall average win rate of 20% against all others. When playing against the Shaman deck, the MCTS shows a maximum win rate performance near 55%. By contrast, when playing against the Zoolock, Hunter, Druid, Warrior, and Paladin, it displays a win rate performance near 20%. The worst observed case was observed when playing against the Paladin deck, where the MCTS received an average win rate near 5%. On the other hand, when using the BC-Greedy, it displays an overall average win rate near 80%. Nevertheless, the worst observed win rate for the BC-Greedy was also when playing against the Paladin deck.

Chart 27: Shaman win rate analysis playing against Monte Carlo Tree Search.

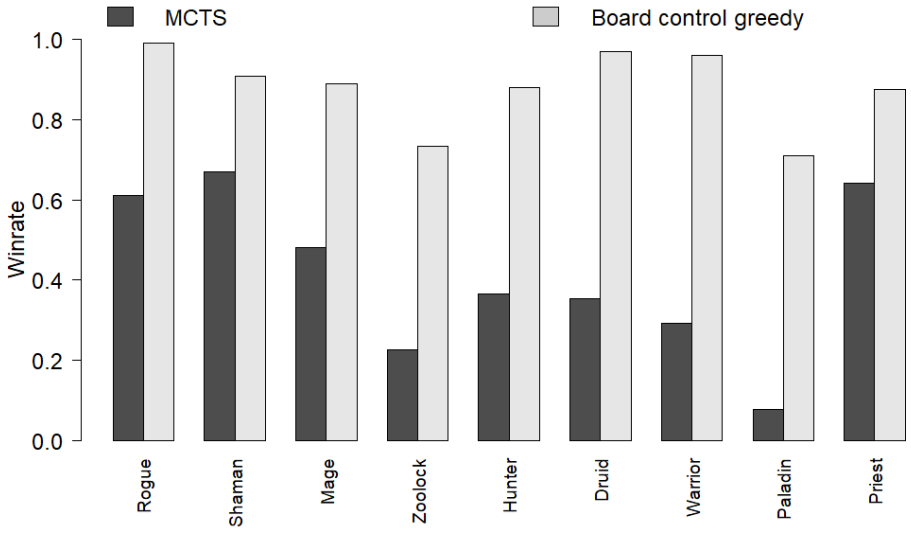


Source: By the Author.

The mage deck performance for the MCTS illustrated in Chart 28, displayed a slightly better performance than the Shaman illustrated in Chart 27. However, it can be

noted that its win rate performance was proportional to the ones observed to the Shaman deck, where the maximum observed average win rate, of 64%, was obtained when playing against the Shaman and the worst observed win rate performance, of 9%, was obtained when playing against the Paladin deck. By contrast, when using the BC-Greedy, it displays a minimum win rate when playing against the Paladin deck.

Chart 28: Mage win rate analysis playing against Monte Carlo Tree Search.



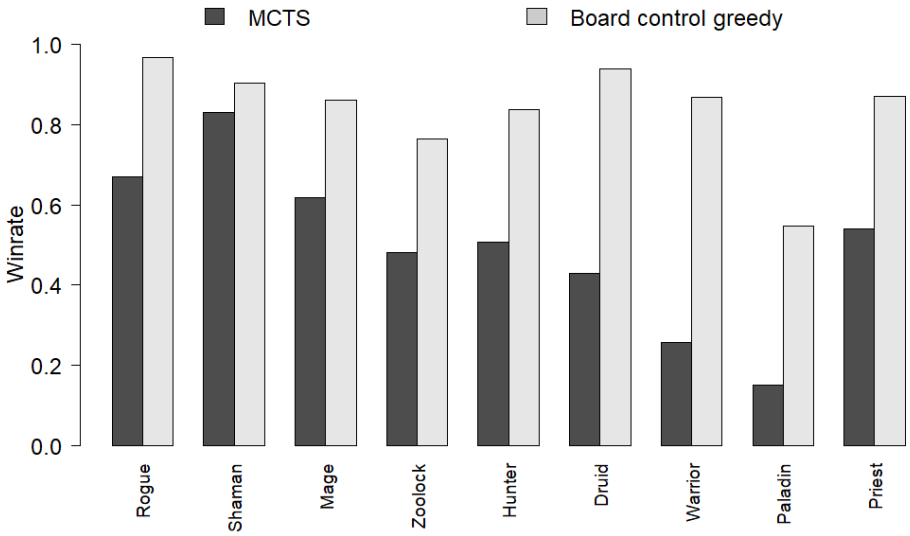
Source: By the Author.

When analyzing the Hunter playing with the MCTS, illustrated in Chart 29, the win rate behavior it can be noted that its performance was better than the observed ones for the Mage and Shaman. It also displays a better performance when playing against the Zoolock, Hunter, Druid, Warrior, and Paladin, where at a win rate difference near 40% was observed. In addition, the worst scenario, when playing with Hunter, was also observed when playing against the Paladin deck. Furthermore, the worst case for the Hunter deck, when playing with the BC-Greedy, was also obtained playing against the Paladin deck.

As showed in Chart 30, the observed average win rates for the Warrior deck playing against a BC-Greedy was worst than the ones obtained from all previously analyzed decks. For the MCTS, it displays a maximum win rate of 13%, against the Hunter deck and minimum of 5% against the Druid deck. For the BC-Greedy, it displays a maximum win rate of 60%, against the Rogue deck and a minimum of 8%, against the Shaman deck.

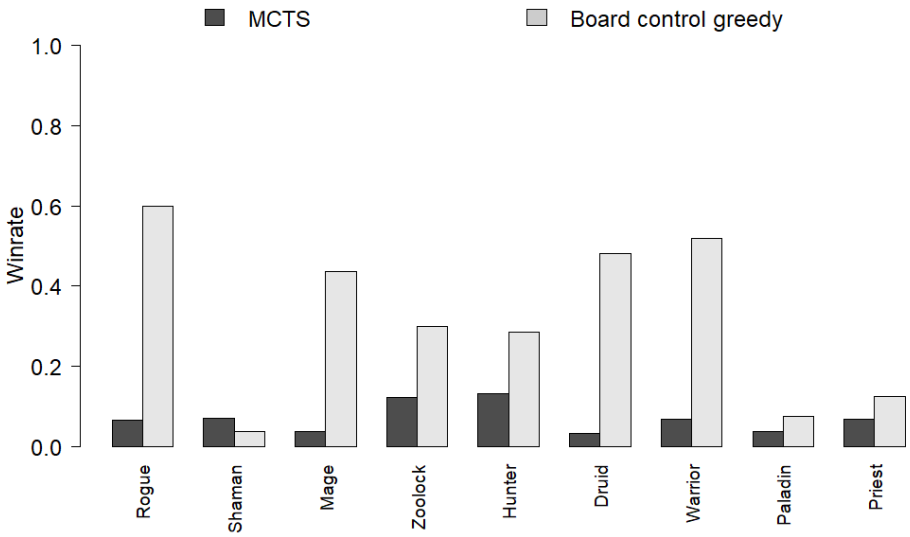
The MCTS playing with the Druid deck, as illustrated in Chart 31, obtained an overall win rate performance against the BC-Greedy near 10% and for the BC-Greedy it scored an overall average of 30%. The maximum win rate for the MCTS, of 14%, was

Chart 29: Hunter win rate analysis playing against Monte Carlo Tree Search.



Source: By the Author.

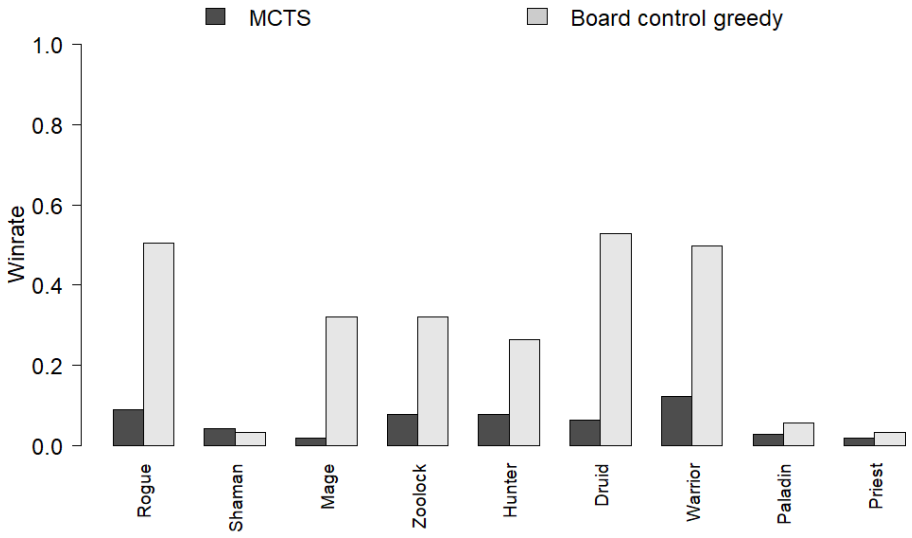
Chart 30: Warrior win rate analysis playing against the Board Control Greedy.



Source: By the Author.

observed when playing against the Druid deck. By contrast, the worst win rate of it, as 2%, was observed when playing against the Priest deck. For the BC-Greedy, the best-observed performance, of 57%, was obtained when playing against the Druid deck and the worst case observed, of 4%, was obtained when playing against the Priest deck.

Chart 31: Druid win rate analysis playing against the Board Control Greedy.



Source: By the Author.

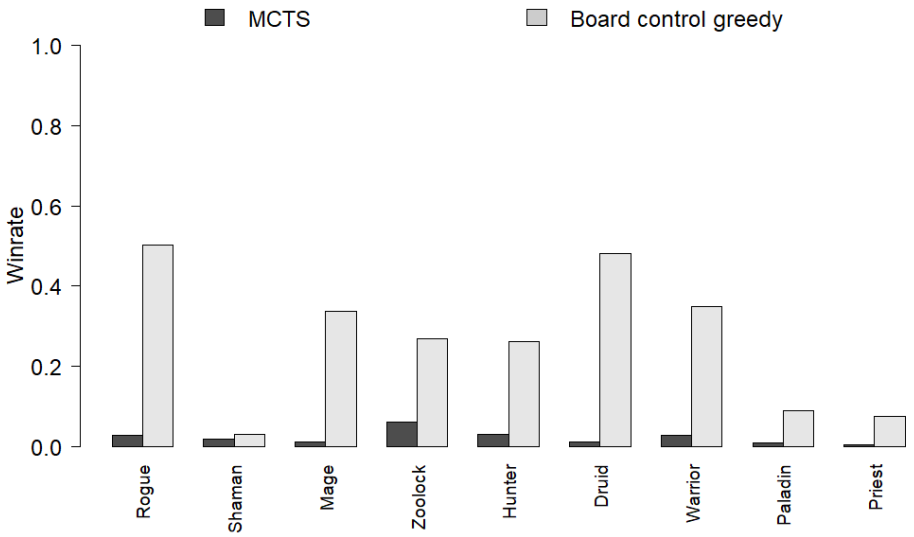
The last individual win rate analysis for deck selection was performed for the Rogue deck. As showed in Chart 32, when playing with the MCTS, it displays an overall average win rate of 4%. By contrast, when using the BC-Greedy it displays an overall average win rate of 34%. The minimum observed win rate for the MCTS was 1%, playing against the Priest deck, and the maximum was observed when playing against the Zoolock deck. For the BC-Greedy, the maximum observed win rate was 51%, playing against the Rogue deck, and the minimum observed win rate, of 4%, was obtained when playing against the Shaman deck.

Based on all aforementioned analysis, the selected decks with their respective enemy were summarized in Table 8.

B.1 Behavioral T-HearthBots winrate

The win rate convergence for all T-HearthBots, assembled using a behavioral action model, was evaluated for all the selected heroes described in Table 8. For the Shaman versus MCTS paladin, as illustrated in Chart 33, T-HearthBot, CTUH-HearthBot, CTH-

Chart 32: Rogue win rate analysis playing against the Board Control Greedy.



Source: By the Author.

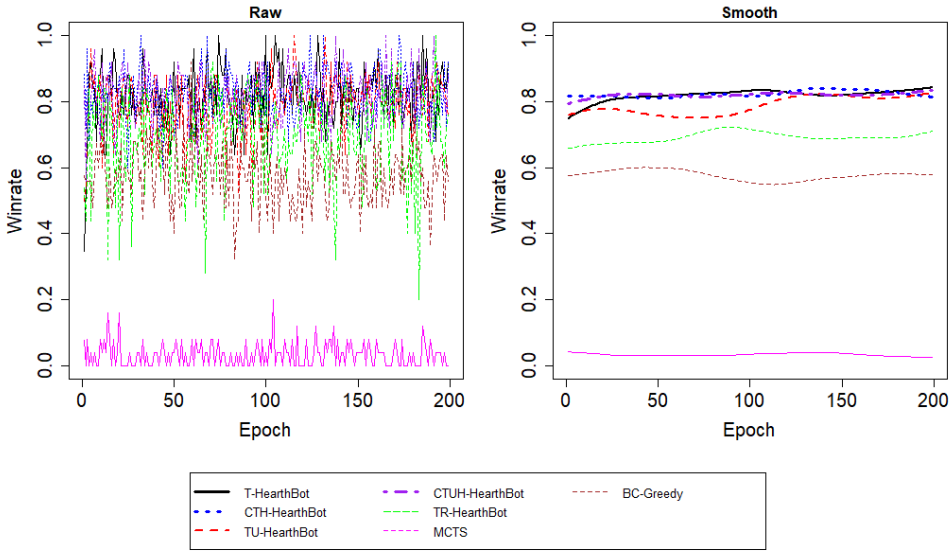
Table 8 – Selected heroes with the respective play style and enemy for all T-HearthBots experiments.

Hero	Play style	Enemy	Enemy Bot
Tempo Mage	Defensive	Secret Paladin	MCTS
Aggro Shaman	Aggressive	Secret Paladin	MCTS
Face Hunter	Aggressive	Secret Paladin	MCTS
Control Warrior	Defensive	Aggro Shaman	Board Control Greedy
Malygos Rogue	Aggressive	Aggro Shaman	Board Control Greedy
Midrange Druid	Hybrid	Control Priest	Board Control Greedy

HearthBot and TU-HearthBot had displayed a similar performance, near 80% win rate, after their convergence. By contrast, the TR-HearthBot obtained an average win rate performance near 65%. The Metastone bots, MCTS and BC-Greedy, obtained an inferior win rate when playing against the MCTS Paladin, where the MCTS win rate can be observed as around 5% and the BC-Greedy win rate around 58%. This result shows that the Shaman deck can achieve high performance when playing with a greedy strategy, since its behavior is most aggressive, that relies on giving as much damage as it can. However, the greedy strategy is not sufficient in order to achieve maximum performance, since the temporal behavior over a Time Flow can influence how good actions will be. Besides TR-HearthBot received a higher win rate than the BC-Greedy, it seems that the one step temporal optimization performed by it, during the virtual POMDP optimization, was not sufficient in order to achieve the same performance as it Q-Learning counterparts.

The summarized behavior of each T-HearthBot when playing with the Shaman

Chart 33: Winrate convergence for Aggro Shaman versus Metastone MCTS playing with Secret Paladin.



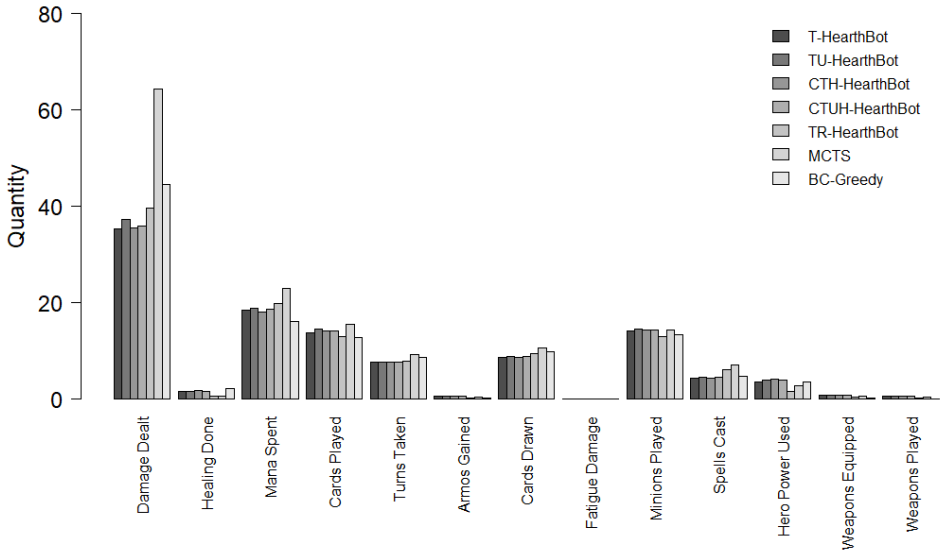
Source: By the Author.

versus MCTS Paladin is showed in Chart 34. As can be seen, T-HearthBot, TU-HearthBot, CTH-HearthBot and CTUH-HearthBot received similar counting for each observed attribute displayed on the X-axis. The main discrepancy occurred when observing the behavior of the MCTS and BC-Greedy. For instance it seems that the MCTS tend to give a high amount of damage if compared to other bots. Besides the observed MCTS damage seems high enough, allowing to win a game, it can not be guaranteed that it was used wisely, since the MCTS rely mostly on the generation of random constructions. Furthermore, the BC-Greedy also displays a higher damage dealt than the other bots and it also had spent less mana. Giving that the win rate convergence, for the Shaman deck for all T-HearthBots, as illustrated in Chart 33, was near 80%, it is possible that the damage dealt and mana spent, if balanced, displays a sign of good performance.

The win rate convergence for the Mage versus MCTS Paladin, as showed in Chart 35, displays a similar performance as the Shaman convergence. The T-HearthBot, TU-HearthBot, CTH-HearthBot and CTUH-HearthBot received a win rate performance near 90% after their convergence. On the other hand, the MCTS received an average win rate performance near 10% and the BC-Greedy received a performance near 75%. It seems that the Mage deck has a better performance than the Shaman one, because it relies in a hybrid strategy, thus allowing to obtain a better performance in various different ways and allowing the MCTS and other bots in exploiting its capabilities.

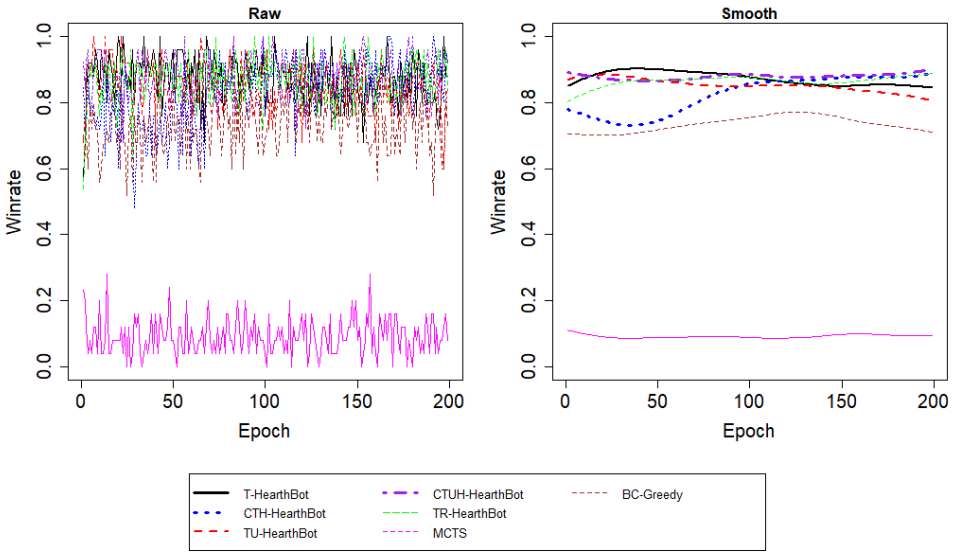
As shown by the behavior analysis, illustrated in Chart 35, the mage Deck relies

Chart 34: Behavior statistics for Aggro Shaman versus Metastone BC-Greedy playing with Secret Paladin.



Source: By the Author.

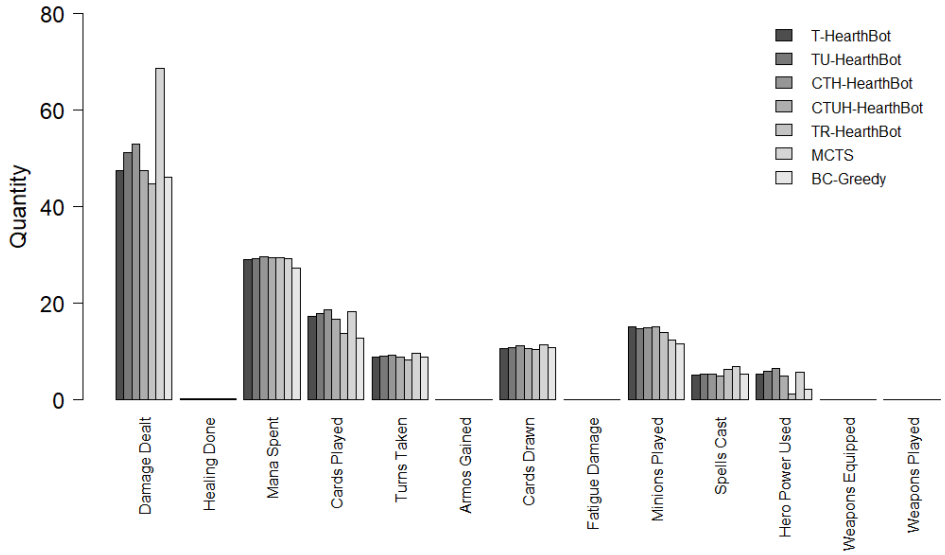
Chart 35: Winrate convergence for Tempo Mage versus Metastone MCTS playing with Secret Paladin.



Source: By the Author.

most on mana managing, differently from the Shaman deck, since the spent mana for all T-HearthBots, including the BC-Greedy, was similar. In terms of damage dealt, the MCTS displays the highest damage done in contrast to the TR-HearthBot that displayed the lowest damage. For all other T-HearthBots, from T-HearthBot to CTUH-HearthBot, the damage done for this deck follows a normal distribution. It can be also observed that the TR-HearthBot tends to not use much of hero power. The observed behavior indicates that the hero power, when playing with the mage deck, can impact hugely the performance of it in terms of win rate, since the mage hero power can help in dealing with dangerous situations, by destroying minions, and also in killing the enemy hero. Furthermore, it seems that a game can be won by managing the damage dealt efficiently.

Chart 36: Behavior statistics for Tempo Mage versus Metastone BC-Greedy playing with Secret Paladin.

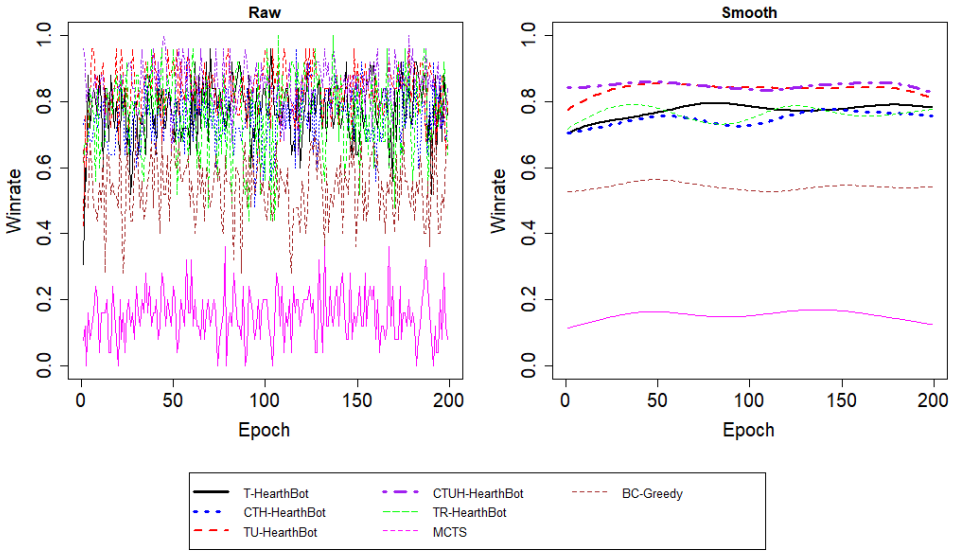


Source: By the Author.

The selected Hunter deck relies in a face strategy, where the best moves for this deck are related in dealing direct damage to the enemy hero. The win rate convergence analysis for this deck, showed in Chart 37, the TU-HearthBot and CTUH-HearthBot received an average win rate near 80%, by contrast, the T-HearthBot, CTH-HearthBot and TR-HearthBot received an average win rate performance near 70%. Moreover, the observed performance for the MCTS was near 15% and for the BC-Greedy was near 53%. The difference in performance, for TU-HearthBot and CTUH-HearthBot between T-HearthBot and CTH-HearthBot, could be due to the fact that the face strategy rely mostly in precisely define the target as the enemy hero that the TU-HearthBot and

CTUH-HearthBot accomplished better. They have accomplished that better, since their resonance for learning on channel 1, action channel, is configured as a 100% for learning and also for prediction.

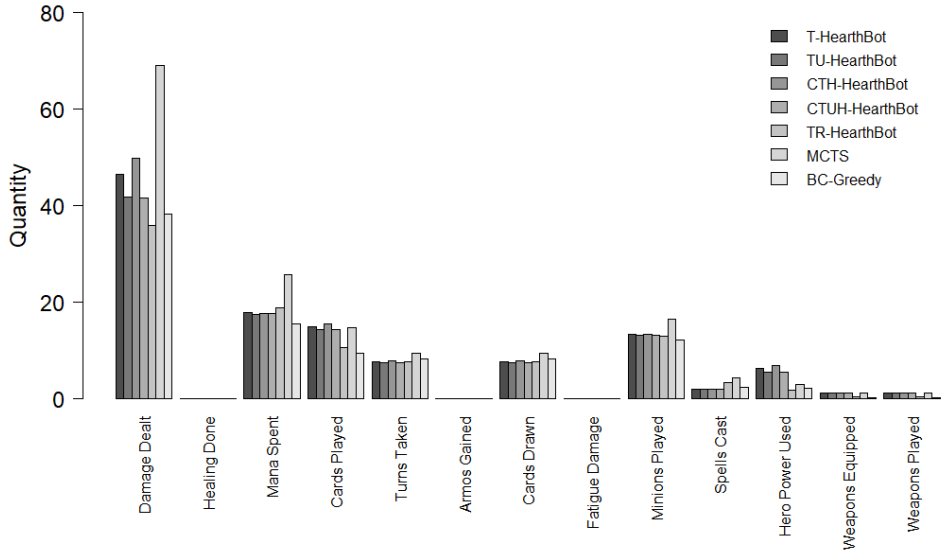
Chart 37: Winrate convergence for Face Hunter versus Metastone MCTS playing with Secret Paladin.



Source: By the Author.

The behavior of the Hunter deck playing against the MCTS Paladin, as illustrated in Chart 38, shows that the MCTS tends to spend more mana by playing more minions and spells. The lowest behavior of the MCTS can be explained as the lack of ability of the bot in deciding efficiently how to expend those resources. Furthermore, as already observed from the previously presented behaviors, the damage dealt does not correlate directly with the win rate performance, since both, T-HearthBot variants and the BC-Greedy, had displayed a low damage dealt, besides that, all T-HearthBots displayed better performance than the BC-Greedy. In addition, it seems that wisely deciding how to spend mana can also influence how good the bot will behave. If analyzing the amount of minions played, can be noted that the MCTS had played the most and for all other bots the quantity of played minions stayed similar. The best performance bot, CTUH-HearthBot, with the fastest convergence for the Hunter experiment, had displayed a higher spell usage with a low damage dealt, same mana spent as other HearthBots and played the least amount of cards. This behavior could indicate that the converged simulated POMDP, obtained through the UAM with 100% resonance in channel 1 for the Hunter deck, tends to use more spells to dominate the battlefield in order to exploit better its face capabilities.

Chart 38: Behavior statistics for Face Hunter versus Metastone MCTS playing with Secret Paladin.

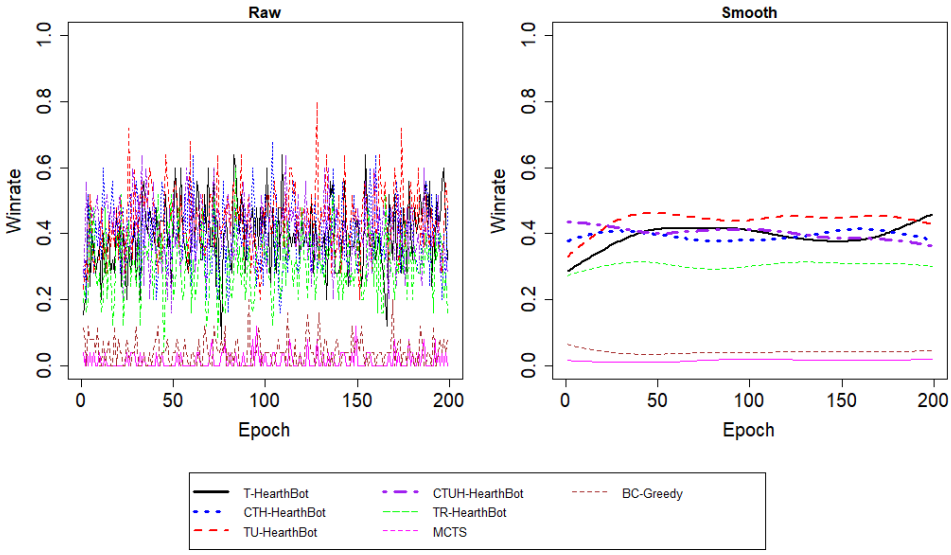


Source: By the Author.

When playing against the BC-Greedy from Metastone, bots tend to have the worst performance, since the BC-Greedy knows exactly how the game will behave in at least one step into the future. As showed in Chart 39, the Rogue deck versus BC-Greedy Shaman displays an average performance, for T-HearthBot, CTH-HearthBot, TU-HearthBot and CTUH-HearthBot, near 40%. The TR-HearthBot had the worst performance near 30% among the other T-HearthBots. This difference in performance could be explained due to the lack of ability by TR-HearthBot in perceive situations ahead from the analyzed sample obtained from a Time Flow. Besides receiving a performance near 40%, all T-HearthBots had performed substantially better than the MCTS, that received an average of 2% win rate, and the BC-Greedy, that received an average of 8% win rate for this deck. The main difference from this experiment from the previously presented ones is related to the fact that the opponent, BC-Greedy, knows exactly how good an action will be one step in to the future, thus allowing it in overcoming its opponents. It can also be argued that the BC-Greedy Rogue received a low win rate average since the Shaman deck was observed as the one with the maximum advantage against the BC-Greedy Rogue, where the expected behavior was a 50% win rate.

Differently from all observed behaviors when playing against the MCTS, the Rogue versus BC-Greedy Shaman behavior, as illustrated in Chart 40, shows that a low damage can, in fact, be able to reduce the performance of the bot, since the MCTS and BC-Greedy possess the lowest damage counting and lowest performance. It can also be noted that

Chart 39: Winrate convergence for Malygos Rogue versus Metastone BC-Greedy playing with Aggro Shaman.



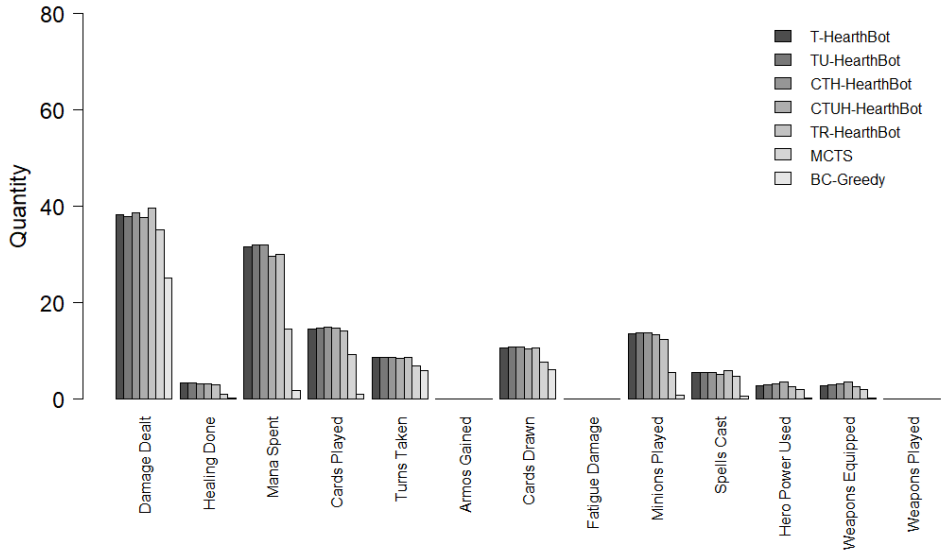
Source: By the Author.

both bots also received the lowest values for all other observed attributes. This behavior can indicate that the handcrafted bots can not handle well a deterministic heuristic that have access to precise information from the Hearthstone environment. For all other bots, its seems that the behavior attributes counting stayed balanced, consequence of using the same temporal technique for learning. It was expected that TR-HearthBot received different, smaller or higher, behavior curve than other bots, since its performance was the worst. However, small variations on its attributes can be noted, specially related to the damage dealt, and mana spent.

As showed in Chart 41, the Druid versus BC-Greedy Priest shows an average win rate performance of 25% for the TU-HearthBot and CTUH-HearthBot. Considering that the Druid deck relies in using precisely its mechanics, related to healing and creating mana crystals, this performance can be explained as the ability of both bots that allow them to predict and categorize precise action information. By contrast, the other T-HearthBot variants received an average win rate performance near 12%, since those bots are not able to categorize the Druid actions precisely. In addition, the MCTS and BC-Greedy received the worst average performance near 4%.

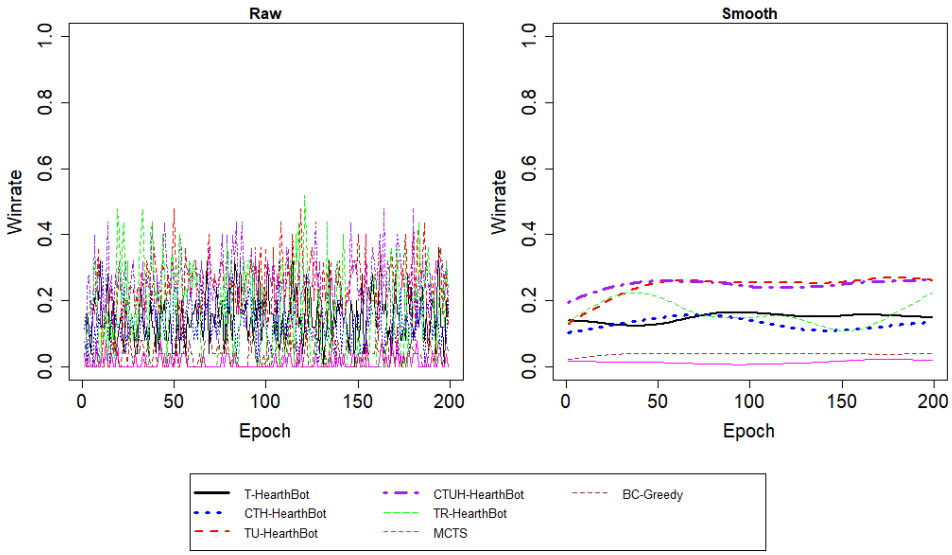
The Druid deck main strategy is to build up mana crystals, control the board, and self heal. As shown by the behavior analysis illustrated in Chart 42, the best bots, CTUH-HearthBot and TU-HearthBot, have dealt different amounts of damage. Assuming that the damage was done wisely, selecting in which minion or hero it would better impact

Chart 40: Behavior statistics for Malygos Rogue versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

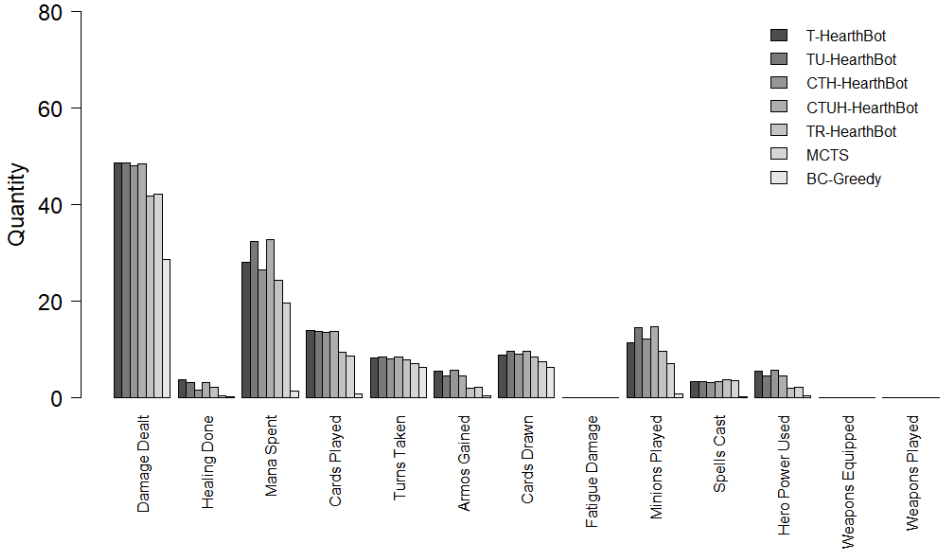
Chart 41: Winrate convergence for Midrange Druid versus Metastone BC-Greedy playing with Control Priest.



Source: By the Author.

the performance of the agent, both performed equally. In addition, their damage was also equal to the ones observed for other bots. A way in expressing the difference in performance between T-HearthBots is through the measurement of mana spent and minions played, since both show discrepancies. On the other hand, the MCTS and BC-Greedy show the worst performance for this case and displayed a low counting for all attributes, because they can not handle the enemy heuristic well.

Chart 42: Behavior statistics for Midrange Druid versus Metastone BC-Greedy playing with Control Priest.

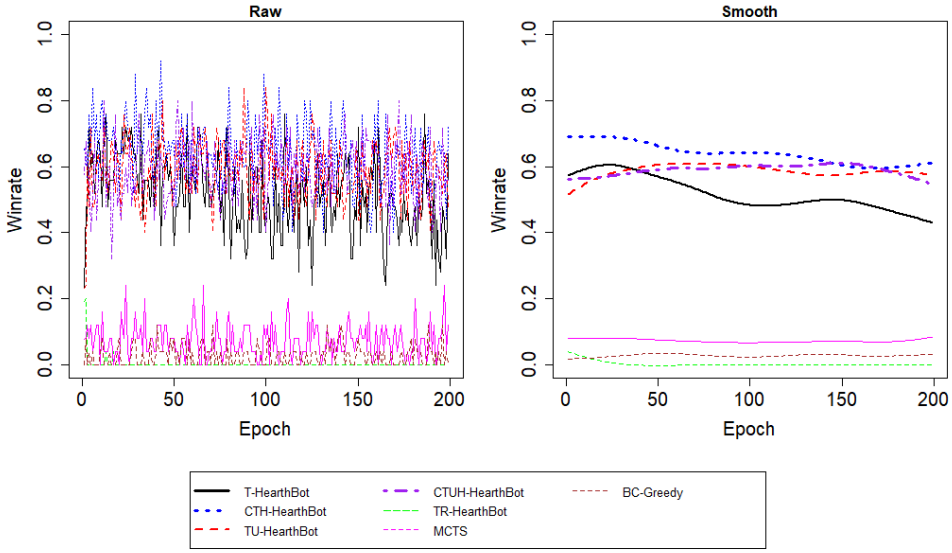


Source: By the Author.

The win rate convergence for the Warrior versus BC-Greedy Shaman is showed in Chart 43. For this experiment, all T-HearthBot variant, excluding T-HearthBot, received an average win rate performance near 60% after their convergence. It is important to note that in this experiment, the convergence of T-HearthBot and CTH-HearthBot follow a negative slope. This behavior can be explained as the lack of ability from those bots in learn Q-values correctly, since both possess the highest number of neurons, because they are not exploiting a tree structure as the TU-HearthBot and CTUH-HearthBot. For this experiment, the TR-HearthBot received a win rate performance, equal to 1%, that is bellow the MCTS. Since the Warrior defensive deck relies mostly in building up the battlefield in order to avoid damage, it seems the TR-HearthBot was not able to optimize the virtual POMDP considering the future, thus it had not exploited well the deck strategy. The MCTS received a win rate performance near 5% and the BC-Greedy received a win rate performance near 10%. Since both are handcrafted methods, this behavior was expected

when playing against the BC-Greedy heuristic from Metastone.

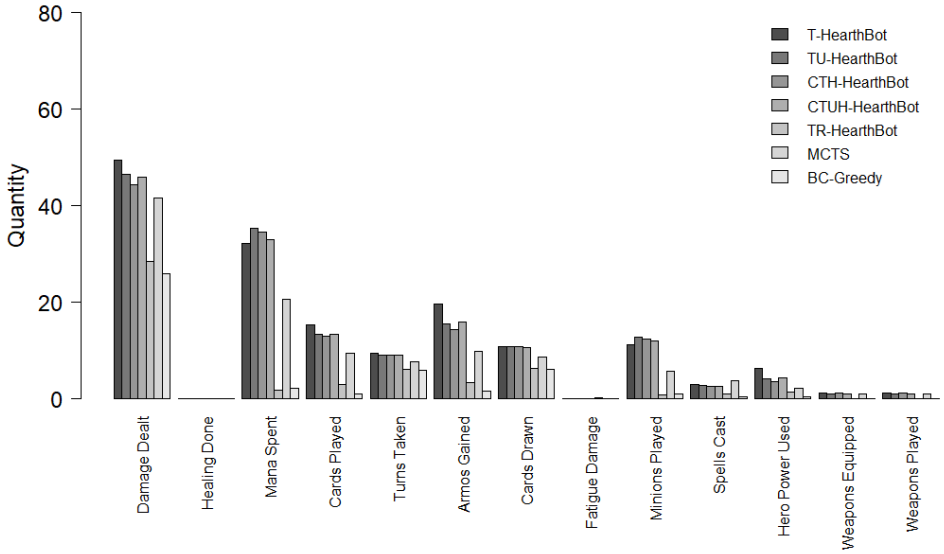
Chart 43: Winrate convergence for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

The behavior of the TR-HearthBot, as illustrated in Chart 44, received the lowest counting for all observed attributes on the X-axis. This reflects, explicitly, on its win rate performance when playing the game, as showed in Chart 43. For all other bots, the counted behavior attributes follow the same pattern, where all T-HearthBots were observed with the same counting and the MCTS and BC-Greedy with relative lower ones. In addition, the Warrior deck strategy, that relies mostly in defending the battlefield and hero with armor skills and defensive minions, reflects directly on the observed armor gained, and minions played that received relatively higher counting if compared to the other decks. It was also observed that this deck allows players in complementing their battlefield by using weapons, thus a counting for weapon usage is also present on its behavior counting illustrated in Chart 44.

Chart 44: Behavior statistics for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



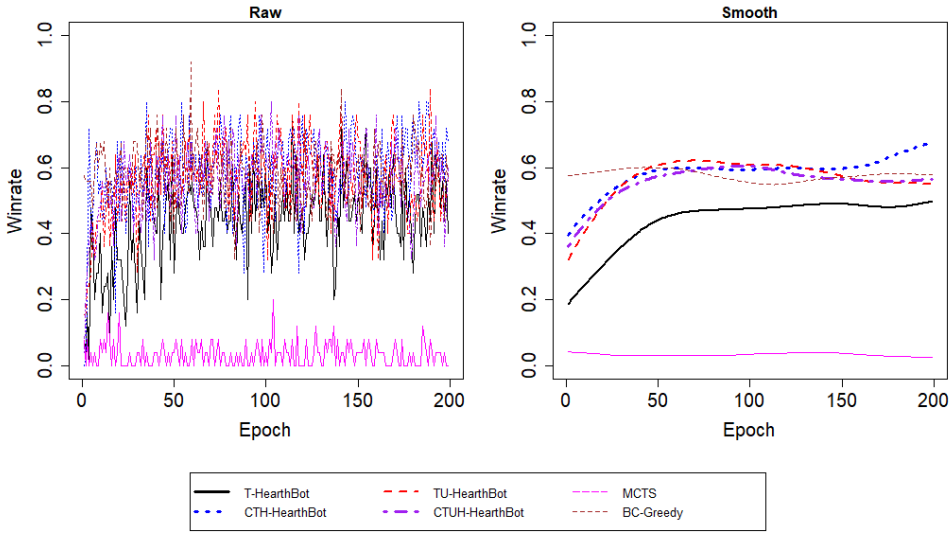
Source: By the Author.

B.1.1 Spectrum T-HearthBots winrate

The result for the Shaman deck playing against the MCTS Paladin shows, for all T-HearthBots variants excluding T-HearthBot, a win rate performance near 60% after convergence. On the other hand, T-HearthBot received a win rate performance near 44%. It was observed a near 15% performance gain if comparing T-HearthBot with all other HearthBots. This behavior can be explained as the ability of TU-HearthBot in precisely categorize and predict actions through its action channel 1. Furthermore, all the creative HearthBots, CTH-HearthBot and CTUH-HearthBot, achieved the displayed performance because the proposed HARP allows better exploration of the search space. If considering the BC-Greedy as a baseline, the creative and UAM HearthBots tend to surpass it for an average of 4% win rate, since the BC-Greedy scored an average of 56%. On the other hand, the MCTS scored worst by receiving an average of 6% win rate during all epochs.

The behavior of the Shaman deck showed in Chart 46, for the spectrum model, differs from the one presented for the behavioral model, since all T-HearthBots show a higher count for all attributes. It is important to note that, the count of each T-HearthBot attribute was higher than the observed ones from the BC-Greedy heuristic. Besides receiving a similar win rate, the BC-Greedy heuristics behavior seems not to be correlated with the observed behavior for all T-HearthBots. The higher damage count for all T-HearthBots can be explained as the ability of those bots in exploiting better the

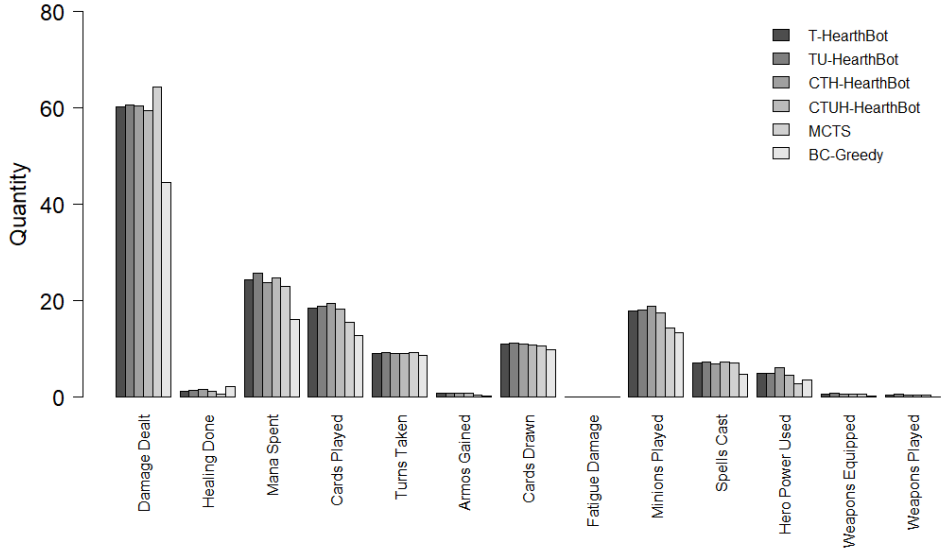
Chart 45: Winrate convergence for Aggro Shaman versus Metastone MCTS playing with Secret Paladin.



Source: By the Author.

Shaman strategy, since it is mainly based in giving as much damage as it can. Considering that the performance of T-HearthBots were slightly better than the one observed for the BC-Greedy, it can be assumed that the temporal exploitation of the POMDP helped in developing such a strategy, from scratch, that allows it to give more damage, thus exploiting well the Shaman deck strategy.

Chart 46: Behavior statistics for Aggro Shaman versus Metastone BC-Greedy playing with Secret Paladin.

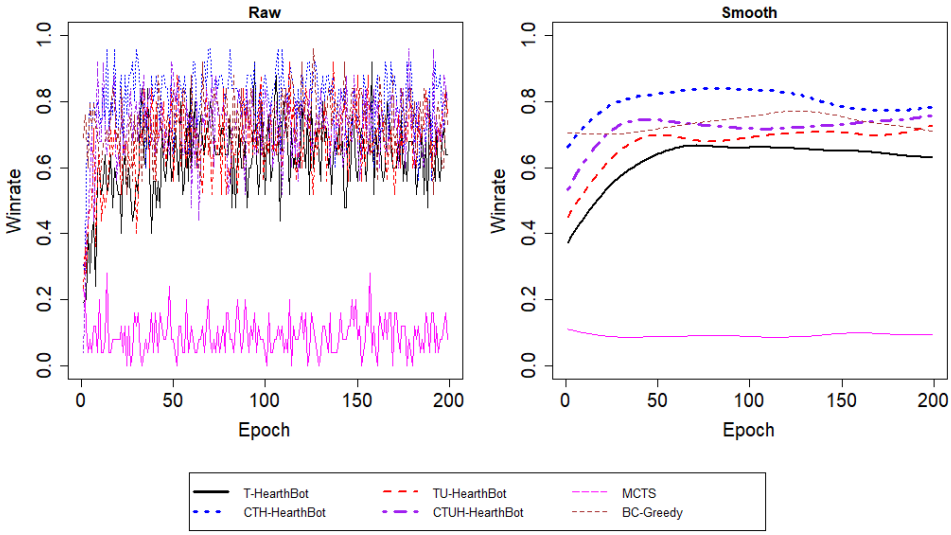


Source: By the Author.

As illustrated in Chart 47, the Mage versus MCTS Paladin shows an average win

rate performance going from 73%, for TU-HearthBot and CTUH-HearthBot, up to 80% for CTH-HearthBot. By contrast, the T-HearthBot received an average win rate performance bellow 65%. These results indicate that the CTH-HearthBot is discovering strategies that are allowing it to develop a better performance. In addition, the CTUH-HearthBot does not show the same win rate average because it relies in precise categorizations in channel 1, thus in order to achieve high performance it needs to categorize a high number of actions during more epochs. Despite CTUH-HearthBot's observed performance being lower than the one observed for CTH-HearthBot, it performed better than TU-HearthBot since it can explore more the environment by using the proposed HARP system. On the other hand, the MCTS received the worst average performance, 10%, and the BC-Greedy received an average win rate near 73%, that is a similar performance as observed for TU-HearthBot and CTUH-HearthBot.

Chart 47: Winrate convergence for Tempo Mage versus Metastone MCTS playing with Secret Paladin.

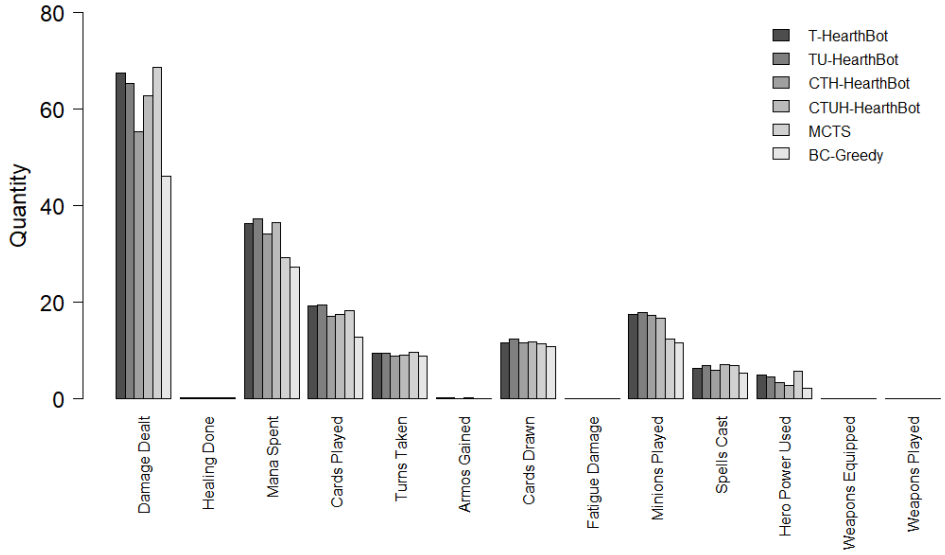


Source: By the Author.

The behavior of the Mage versus the MCTS Paladin, as showed in Chart 48, shows that the three best performance boots, TU-HearthBot, CTH-HearthBot and CTUH-HearthBot, have dealt the least damage if comparing to tall other bots. In addition, it seems that the CTH-HearthBot and BC-Greedy have given the least amount of damage, since they apparently better manage their damage dealing capabilities. The observed behavior for this deck was different from the Shaman, since its strategy relies in board control, thus the better performance of it can be related to damage control, mana managing,

cards played and Hero Power usage. The best bot displayed the lowest counting for almost all attributes, as shown on the X-axis, thus its behavior can be explained as a developed ability to manage those attributes. By contrast, the worst bot, MCTS, displayed the lowest counting for each attribute, thus a lower counting is not necessarily a synonym of good performance.

Chart 48: Behavior statistics for Tempo Mage versus Metastone MCTS playing with Secret Paladin.

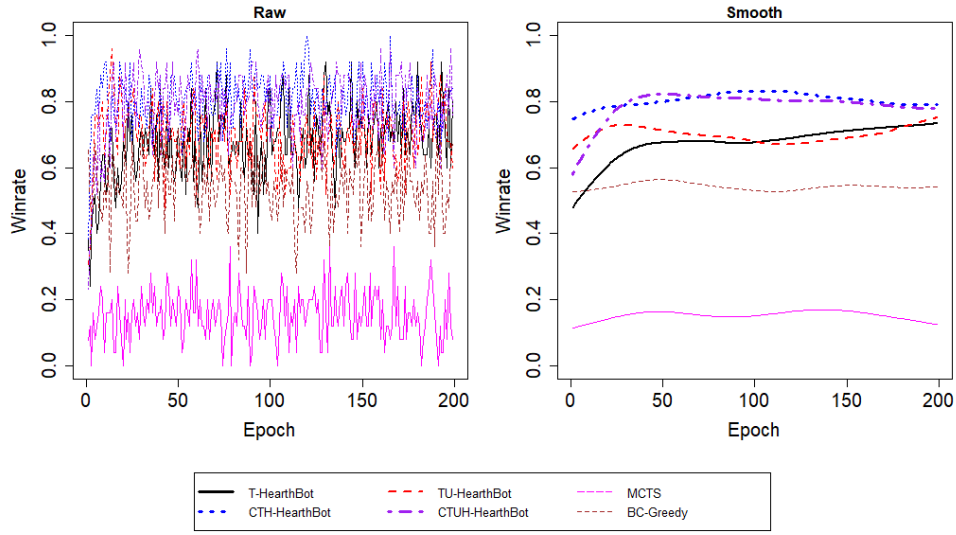


Source: By the Author.

When analyzing the win rate convergence of the Hunter versus MCTS Paladin, illustrated in Chart 49, it can be noted that the creative bots obtained the best performance again, where the CTH-HearthBot and CTUH-HearthBot obtained an average win rate performance of 80%. The TR-HearthBot and T-HearthBot obtained an average win rate performance of 68%. This difference in performance could be explained as the lack of capability, from the non-creative bots, in exploring the search space on an earlier exploration stage, before converging. Furthermore, all T-HearthBot variants received a higher win rate than all analyzed Metastone agents. For instance, the BC-Greedy received an average win rate of 56% and the MCTS received an average win rate of 17%, which indicates a performance gain higher than 50%, if comparing all T-HearthBots with the MCTS, and a performance gain higher than 20%, if comparing with the BC-Greedy.

The Hunter deck displays a behavior similar to the one observed for the Mage deck, as showed in Chart 50. All the creative HearthBots, CTH-HearthBot and CTUH-HearthBot, displayed a lower attribute counting for the damage dealt, cards played and

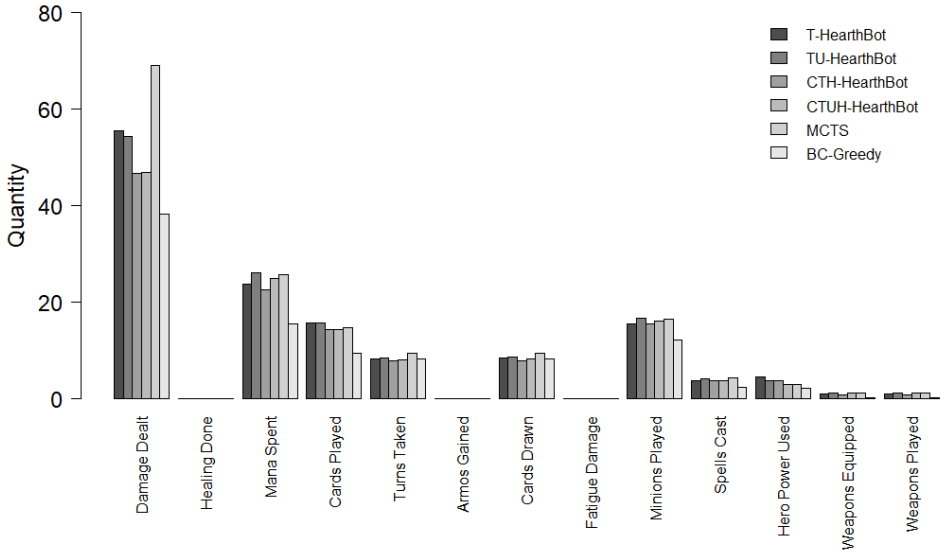
Chart 49: Winrate convergence for Face Hunter versus Metastone MCTS playing with Secret Paladin.



Source: By the Author.

cards drawn. Those counted attributes represent the ones responsible in guiding the deck's strategy, where managing those will possibly increase an agent's performance. As also depicted in Chart 50, the attribute that exhibits the highest counting was the damage dealt, for the BC-Greedy. It could indicate that the BC-Greedy heuristic, as discussed on the previous presented behavior analysis, tries to give the highest amount of damage as possible without caring about how much mana it spent.

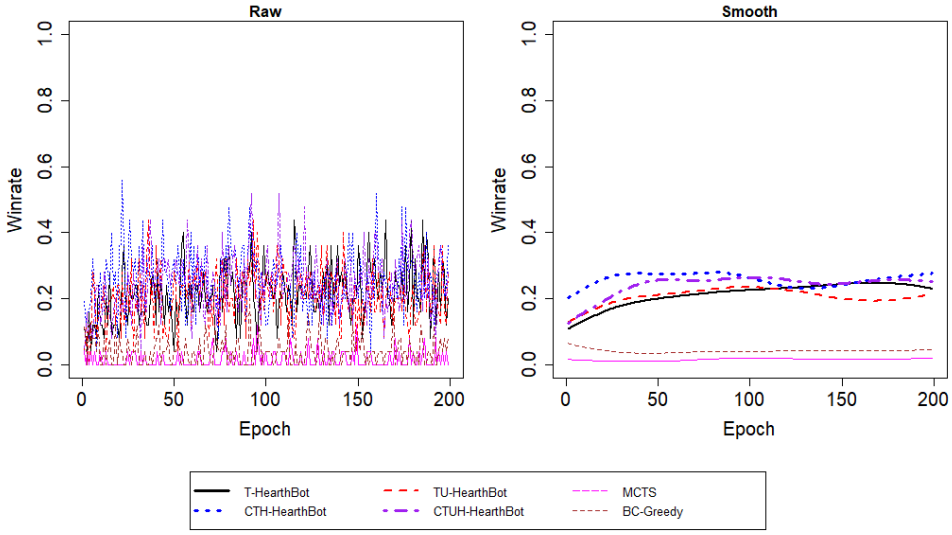
Chart 50: Behavior statistics for Face Hunter versus Metastone BC-Greedy playing with Secret Paladin.



Source: By the Author.

A huge loss in performance, for all bots that use the Hunter deck, was observed when playing against the BC-Greedy Shaman, as illustrated in Chart 51. The CTH-HearthBot and CTUH-HearthBot had displayed an average win rate performance of 27%, while the T-HearthBot and TU-HearthBot had displayed an average performance of 20%. An average performance gain of 7% was observed if comparing the creative bots with non-creative bots for this experiment. Furthermore, the BC-Greedy displayed an average performance of 5% and the MCTS displayed an average performance of 2%. The T-HearthBots for this experiment seems lower than all previous presented ones although they are, in average, 20% higher than the ones obtained for the handcrafted bots from Metastone. Despite the 7% win rate difference, from creative to non-creative bots, a slightly performance loss was observed between the epoch 100 and 200 for all creative T-HearthBots. The observed behavior could be explained as a Q-value corruption that eventually happened inside the network's neurons along its training.

Chart 51: Winrate convergence for Malygs Rogue versus Metastone BC-Greedy playing with Aggro Shaman.

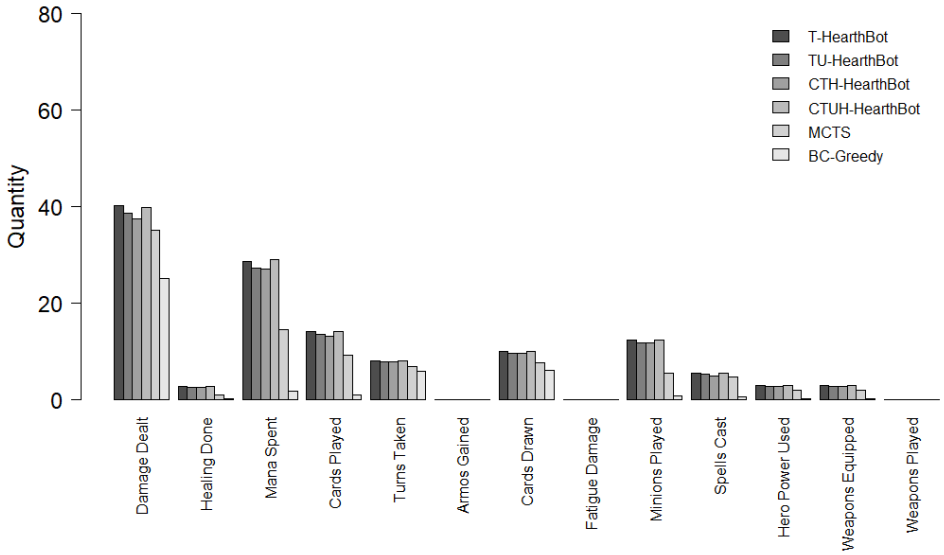


Source: By the Author.

The behavior of the Rogue versus BC-Greedy Shaman, as showed in Chart 52, was different from all observed ones for the spectrum model, since all T-HearthBots displayed a higher attribute count than the MCTS and BC-Greedy from Metastone. As illustrated in Chart 52, the MCTS and BC-Greedy received the lowest value for all attributes as observed also for the other decks. Which respect to the damage dealt, it seems that all T-HearthBots have converged into a similar solution that exploits a certain level of

damage that is distributed among the game components. It can not be inferred in which component all those damage was inflicted on, thus the difference on performance could have be caused by variations on the developed strategy in each bot. In addition, it seems that the BC-Greedy bot exploits the Rogue strategy in a way that allows it to spend the lowest amount of mana as possible, by consequence it plays fewer cards during the game. This can explain its bad performance, because the Rogue deck relies in playing as much cards as it can in order to activate their effects properly.

Chart 52: Behavior statistics for Malygos Rogue versus Metastone BC-Greedy playing with Aggro Shaman.

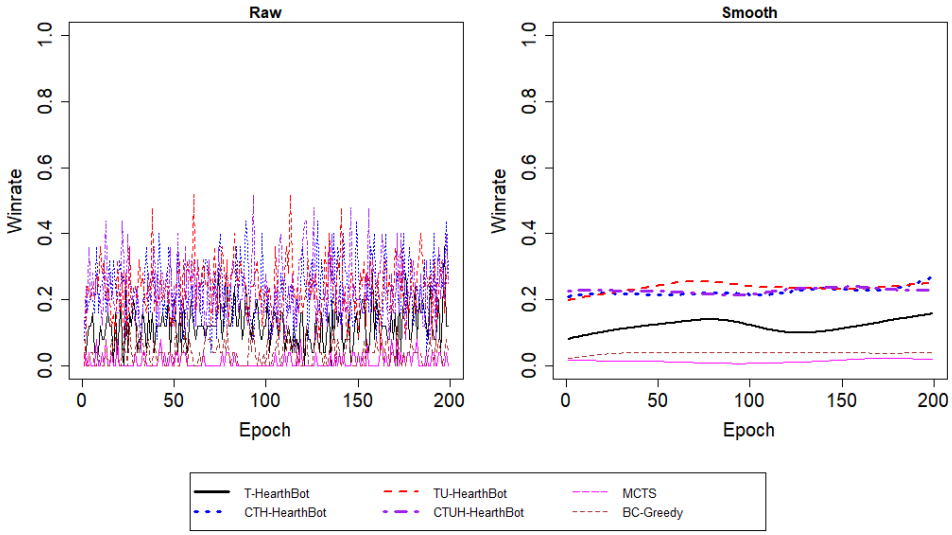


Source: By the Author.

The Druid versus BC-Greedy Priest win rate performance, showed in Chart 53, shows a similar behavior than the one observed for the Rogue deck. It was observed an average win rate performance of 21% for all T-HearthBots proposals excluding T-HearthBot itself. The major point that can be observed in this experiment is the clear impact of the proximity metric used to categorize actions, where it is used inside CTH-HearthBot, TU-HearthBot and CTUH-HearthBot. Due to the large amount of the abilities that the selected Druid deck can perform, the T-HearthBot performed poorly, 10% win rate average, since it can not categorize correctly what actions it should perform in order to maximize its win rate on the virtual POMDP. In addition, the MCTS and BC-Greedy performed in a similar way, where the MCTS obtained an average win rate performance of 3% and the BC-Greedy obtained an average of 5%.

As illustrated in Chart 54, when playing with the Druid versus BC-Greedy Priest,

Chart 53: Winrate convergence for Midrange Druid versus Metastone BC-Greedy playing with Control Priest.

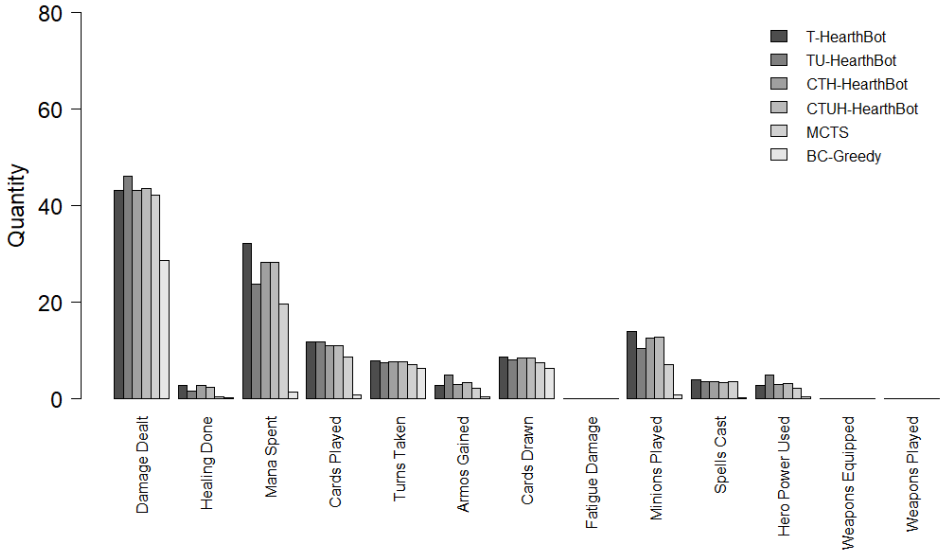


Source: By the Author.

the counted values for all attributes were higher for all T-HearthBots. When looking at the damage dealt and mana spent, it seems that TU-HearthBot displayed a discrepancy, where it received the highest damage count and the third lowest mana spent count. This behavior occurred as mere convergence of the network's Q-values, where it prioritized giving damage and spent less mana. This behavior could occur for each HearthBot, however it does not mean that it represents a better performance. In general, the observed behavior for all T-HearthBots indicates that, when playing with the Druid deck, healing is important, since they displayed a slightly higher healing count during the games. By contrast, the MCTS and BC-Greedy received the lowest counting for each attribute, which indicates that they are unable to explore well the strategy of the Druid deck when playing against the BC-Greedy Priest.

The win rate convergence for the Warrior versus BC-Greedy Shaman, showed in Chart 55, shows that all creative HearthBots displayed an average performance of 50%. In addition, the TU-HearthBot achieve a similar average performance, of 44%, than its creative counterparts. Furthermore, it seems that the TU-HearthBot scored a lower average win rate since it did not explore well during the exploration phase before converging. On the other hand, the T-HearthBot achieved an average win rate of 30%, what leads to an average win rate gain, for the creative HearthBots, of 20%. When analyzing the MCTS and BC-Greedy, it can be seemed that their performance where considerable worst than all the proposed T-HearthBots, where the MCTS achieved an average win rate of 3%

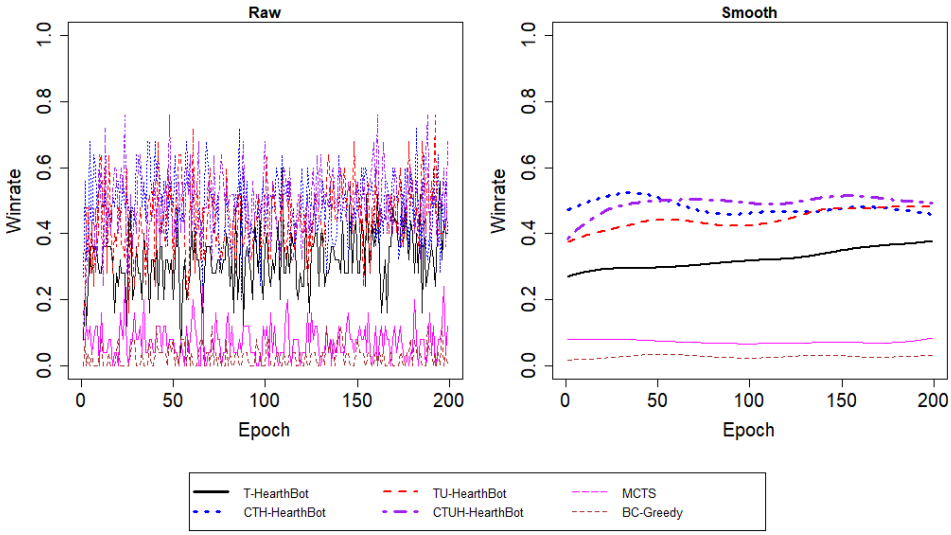
Chart 54: Behavior statistics for Midrange Druid versus Metastone BC-Greedy playing with Control Priest.



Source: By the Author.

and the BC-Greedy achieved an average of 9%. Since the main strategy of the Warrior is on defending its hero through defensive minions and spells, it was expected to observe that performance for all the handcrafted bots from Metastone that can not deal well with temporal information.

Chart 55: Winrate convergence for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.

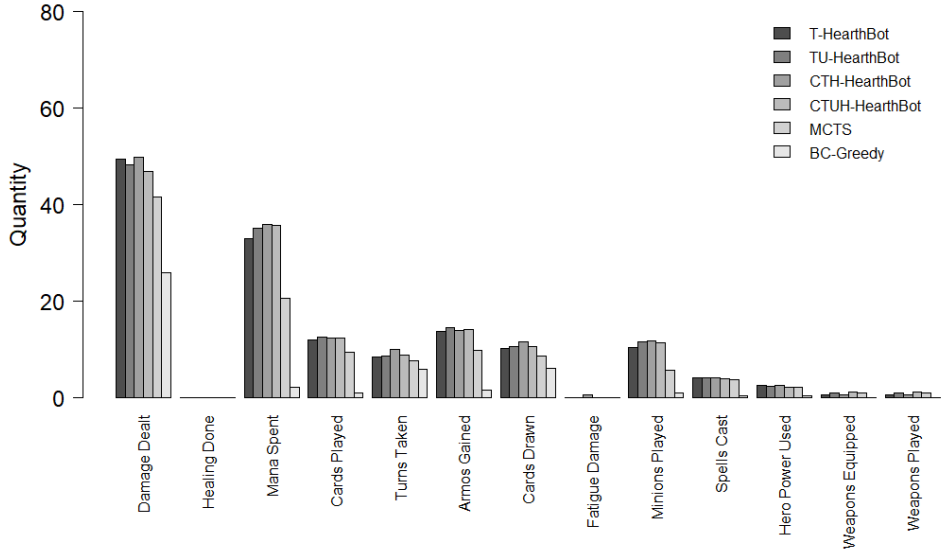


Source: By the Author.

As illustrated in Chart 56, the Warrior versus BC-Greedy Shaman displayed a behavior similar to the one observed for the Rogue versus BC-Greedy Shaman. However,

the presented behavior in Chart 56 does not show discrepancies that can be interpreted as a sign of good or bad performance. All attributes were quite similar for all T-HearthBot, but it can be seemed that the T-HearthBot has spent less mana and it also played less minions. This behavior can explain its poor performance in comparison to other T-HearthBots although all other T-HearthBots played nearly identically. The MCTS and BC-Greedy displayed in this experiment the lowest performance, as showed in Chart 56 and this behavior seems to happen recurrently.

Chart 56: Behavior statistics for Control Warrior versus Metastone BC-Greedy playing with Aggro Shaman.



Source: By the Author.

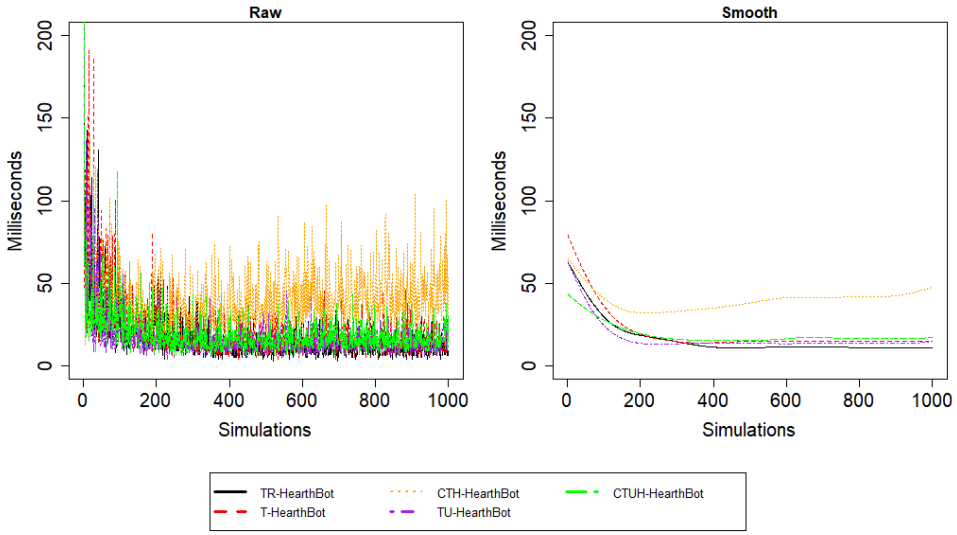
APPENDIX C – PERFORMANCE AND COMPLEXITY ANALYSIS

This appendix presents the performance and complexity analysis only for T-HearthBots, and addressing the performance for HearthBot, since HoningStone is an adapted GRASP procedure that was deeply studied by (BINATO; OLIVEIRA; ARAUJO, 2001) and (FEO; RESENDE, 1995) and HearthBot works in a unique way than its temporal counterparts. The performance of the proposed HearthBot and T-HearthBot systems was evaluated in terms of milliseconds expended by Metastone to complete a simulation. That measurement was performed under a 1000 simulations to preserve the estimated coefficient of variation addressed at Section 11.2. All conducted experiments were accomplished for the Warrior versus Warrior MCTS match, thus lowering the execution time of the simulations within one match.

The execution time of all T-HearthBots was compared to T-HearthBot, because it is the baseline deployed with the original FALCON architecture. In addition, the simulation time was measured considering the entire simulation, because it allows verifying the behavior of Metastone or a game itself when the proposed systems are acting inside of it, thus being more close to reality. In order to check a system, specific expected performance was opted in performing a time complexity analysis, because it demonstrates the asymptotical behavior of the agents alone. The performance for HearthBot is not compared directly with its temporal counterparts because the system is not from the same nature, in terms of functionality, and it explores and exploits routines work in a different way than the ones used by T-HearthBots.

As depicted in Chart 57, it shows the execution time for all behavioral HearthBots, TR-HearthBot displayed an average execution time of 10.82 ms. A similar execution time, of 11.06 ms, was observed for T-HearthBot, for TU-HearthBot, 13.81 ms, and for CTUH-HearthBot, 15.06 ms. On the other hand, CTH-HearthBot displayed an average execution time near 54.45 ms. Nevertheless, it seems that the behavioral model displays a logarithmic decay in execution time, where almost all bots reached an average of 10 ms. Furthermore, the CTH-HearthBot displayed a different performance than the others. This unique performance could be explained as the ability of the HARP system in better exploring the environment, consequently, it will learn more neurons and demand more time to process all of them. However, by that argument, the CTUH-HearthBot had displayed the same behavior, what did not happened.

Chart 57: Simulation execution time for the behavioral HearthBots.

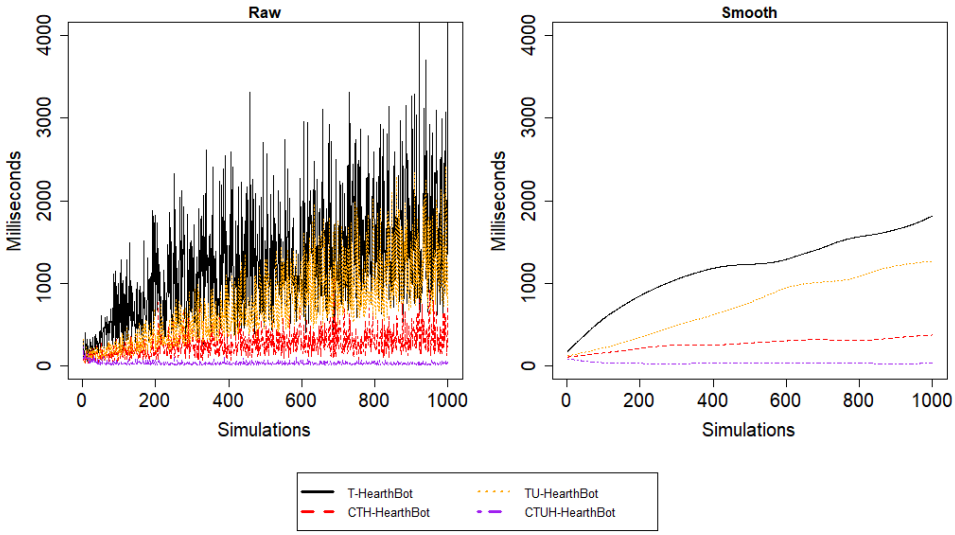


Source: By the Author.

There is also an early decay that happened for the behavior model, depicted by Chart 57, because during that time the system is in exploration mode, thus a large amount of learning different stimulus at the beginning that is gradually decaying until ϵ reaches 0. This decay can also be explained as a convergence of the network with respect to the total amount of useful POMDP states that it codes, thus when reaching the exploitation phase it will stop learning and performing operations inside its structure. Nevertheless, no speedups were observed.

Differently, from the behavioral model, the spectrum one as depicted in Chart 58, displayed a logarithmic growth behavior, that was expected since the action search space for the spectrum model is bigger than the one represented by the behavioral model. The T-HearthBot, for this experiment, displayed an average execution time of 1700.81 ms, what is a huge performance loss if comparing to the behavioral model. However, it should be considered that the spectrum model learns from scratch, thus it is expected that it uses more space, as argued in Section 11.4.6, reducing its performance. In addition, the TU-HearthBot displayed an average execution time of 1239.11 ms, thus giving a speedup of 1.37 over the baseline. On the other hand, CTH-HearthBot displayed an average execution time of 363.41 ms, what is a huge performance upgrade, with its speedup equal to 4.68 if comparing to the baseline. The gain in performance for the CTH-HearthBot can possibly be due to its ability in economize neurons on early stages of development as argued in Section 11.4.6. Finally, the CTUH-HearthBot displayed the best performance in terms of execution time, equal to 26.69 ms, what leads to a speedup of 63.72.

Chart 58: Simulation execution time for the spectrum HearthBots.



Source: By the Author.

The proposed T-HearthBot incorporates an ANN that allows to simulate a POMDP and incorporates the Q-Learning temporal learning technique. The time complexity of its algorithm, as deployed for this research, is represented in terms of the total amount of fields f, each field size t and the total number of neurons n. In order to represent its complexity, it is considered that each field have equal size. The T-HearthBot's algorithm relies in using the fuzzy ART I, fuzzy ART II and, in its variants, the Proximity neuron activation functions. The estimated complexity for calculating each neuron activation function is given as $O(t)$, where this upper bound was calculated considering operations performed within a filed. Furthermore, in order to set the activity for each possible field, the complexity turns into $O(f \times t)$. This same complexity is given also to its readout operation. On the other hand, the perfect mismatch operation, from the FALCON architecture, is given as $O(1)$. In addition, the neuron pruning and neuron decay posses a complexity equal to $O(n)$, while the action selection, for the reactive model, posses a complexity equal to $O(f)$. The main routine, from T-HearthBot's algorithm, is called prediction, and its complexity is presented as,

$$O(n \times f \times t) + O(nlogn) + O(n) + O(f \times t) \tag{60}$$

where $O(n \times f \times t)$ represents the neuron activation function through the composite operation for each neuron and each field, $O(nlogn)$ represents the sorting algorithm, provided by default on Java programming language, used to allow in fast selecting neurons

from the resonance checking procedure, $O(n)$ represents the total number of neurons that can be accessed from the resonance checking procedure and $O(f \times t)$ the readout operation. On the other hand, HearthBot posses the time complexity of $O(f \times t) + O(nlogn)$, where $O(f \times t)$ is the neuron activation function distributed among the GPU cores and $O(nlogn)$ the search algorithm used to compute the max operation from the resonance checking output.

The complexity for all T-HearthBots is expressed in terms of total amount of actions a that they can perform. Assuming that, the complexity of the local reasoner is given as $O(a)$, since it will seek for the best action one by one. Furthermore, the deployed Q-Learning methods are divided into three parts, the learning, exploring and exploiting. The complexity of the learning process is given as,

$$O(a \times n \times f \times t) \tag{61}$$

where $O(a \times n \times f \times t)$ represents the action selection for the learning process. Next, the complexity of the exploring part is given as,

$$O(n \times f \times t) + O(f \times t) + O(t) \tag{62}$$

being $O(n \times f \times t)$ the prediction process, $O(f \times t)$ the set activity and $O(t)$ the performed readout for the reward field. On the other hand, the exploiting process complexity is given as follows,

$$O(a \times n \times f \times t) + O(a \times f \times t) + O(a \times t) \tag{63}$$

where $O(a \times n \times f \times t)$ represents the complexity for performing predictions for each possible action, the $O(a \times f \times t)$ represents the set activity for each action and $O(a \times t)$ the readout for the reward field. Differently, the reactive model uses an action mask and the neuron dynamics described for the reactive model. The complexity of the reactive model, assuming the discussed model, is given as,

$$O(a \times n \times f \times t) + O(a \times t) + O(2 \times f \times t) + O(2 \times t) + O(2n) \tag{64}$$

where $O(a \times n \times f \times t)$ represents the action selection process through predictions, $O(a \times t)$ being the complexity for action selection into a mask composed by t variables, $O(2 \times f \times t)$ the performed set activity for the evaluated environment, $O(2 \times t)$ being the performed readouts from all predictions and $O(2n)$ the pruning and decay process. All evaluated complexities were calculated without adding the local reasoner term, that is $O(a)$, since it is not part of the main routines of the algorithm. Other routines, deployed as $O(1)$, such as the neuron reinforcement and neuron erosion, are not considered either.

The time complexity for the Bayesian surprise was calculated in terms of total amount of elements b, used to represent all past experiences perceived by the agents, and the total amount of coded features v, used by the surprise vectors inside the Expectation ART. In order to achieve the Bayesian surprise, in the worst case, represented as a Bayesian surprise matrix, the mean, sum, and deviation of a population must be calculated. The complexity to calculate such statistics are given as follows,

$$O(b) + O(2 \times v) + O(b \times v) \tag{65}$$

where $O(b)$ represents the total amount of perceived environments, $O(2 \times v)$ is the cost of calculating the mean and variance and $O(b \times v)$ the cost of the algorithm for traversing the perceived features for each experienced environment. The Bayesian surprise, on the other hand, is calculated with the cost,

$$O(v) + O(b) + O(2 \times v) + O(b \times v) \tag{66}$$

where $O(v)$ is the total amount of features, $O(b)$ the total amount of observed environments and $O(2 \times v) + O(b \times v)$ the complexity for updating the Bayesian surprise matrix. By contrast, the deployed surprise on the HARP is calculated with the cost of $O(1)$, since it directly accesses the action counter for any possible action.

On the other hand, the time complexity for the HARP is the same as the one described by Equations 61 and 63. However, the exploration phase is performed by the Expectation ART, thus its complexity is given as,

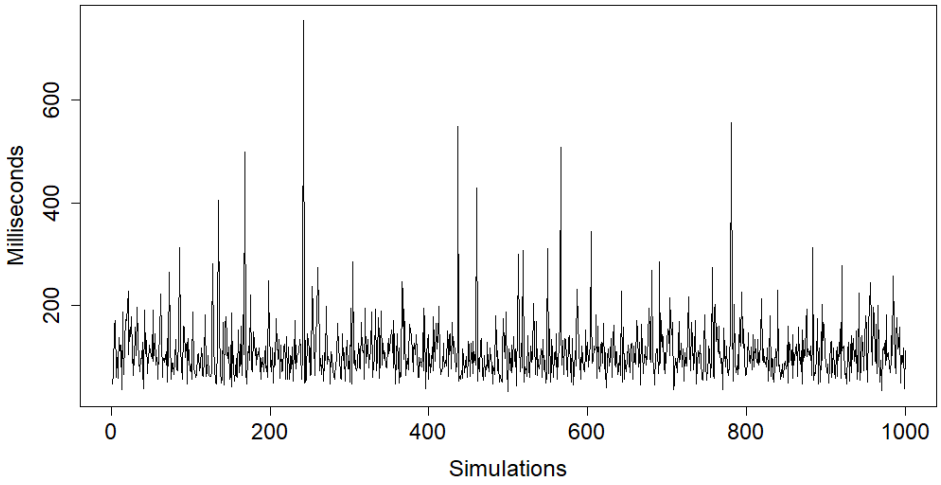
$$O(a \times f \times t) + O(a \times n \times f \times t) + O(t) + O(a) \tag{67}$$

where $O(a \times f \times t)$ is the set activity function and $O(a \times n \times f \times t)$ being the prediction process, $O(t)$ represents the readout operation, $O(a)$ the total amount of actions. The surprise for each action is calculated through the proposed T-HearthBots surprise with complexity equal $O(1)$.

The deployed UAM, with two channels, where channel 1 represents the action field from a FALCON architecture and channel Y representing the environment-reward mapping, posses the time complexity for its channel 1 equal to $O(1)$. That stems from the fact that, the action channel was configured with 100% resonance, thus it could be deployed as an indexed table with constant access time. For the channel Y, the time complexity equals to $O(n \times t)$, where n is the total number of neurons and t the total number of variables used in that channel. By using this configuration, the time complexity was reduced by the total number of fields. Furthermore, the general path activation complexity for the UAM is $O(logn)$ since it was assembled as a balanced tree structure.

By contrast to the observed behavior for T-HearthBots, HearthBot possesses the time complexity equal $O(f \times t)$ distributed among all neurons, since it was deployed into a GPU, and it also possesses $O(n) + 2 \times O(1)$ to perform readout on the activated neuron, where $O(n)$ is used to find the maximum element from the activation and $O(1)$ to perform the set activity and readout operations. That stems from the fact that, HearthBot calculates the resonance of neurons at the same time it is calculating their activation. In addition, the performance time of HearthBot when performing is depicted in Chart 59, where it displayed an average of 111.01 ms. Its performance was constant since the Tesla GPU possesses more than 4000 cores, thus HearthBot is able to handle 4000 neurons running in parallel for the deployed implementation.

Chart 59: Simulation execution time for HearthBot.



Source: By the Author.

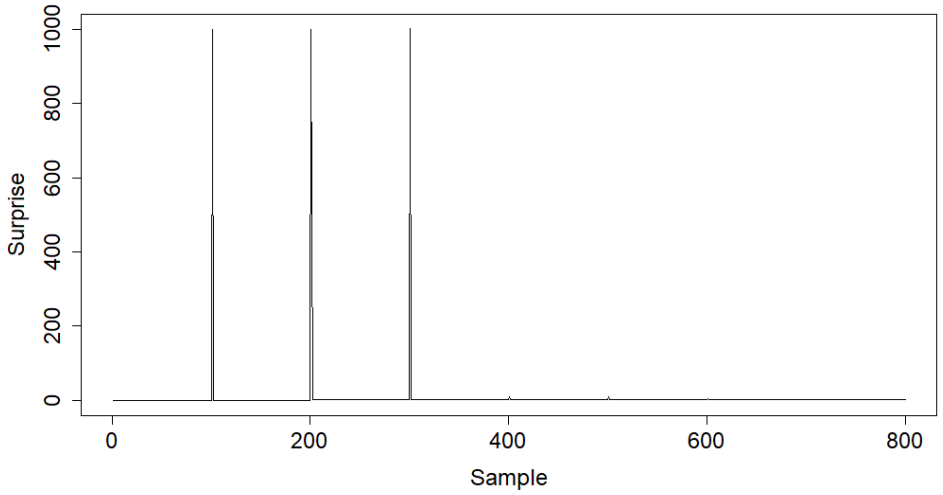
APPENDIX D – BAYESIAN SURPRISE BEHAVIOR

In this appendix is presented a behavior analysis for the deployed surprise model on all HARP based HearthBots. This analysis shows a comparison on how the Bayesian surprise behaves for the HARP system and how it should behave in order to allow compute the correct surprise considering the proposed Hearthstone models. The experiments conducted for this test were made with a sample base composed by 800 3-bit binary numbers, where the base is composed of 8 numbers, each possible permutation for 3-bit representing the decimal numbers from 0 to 7. In order to train the Bayesian surprise model, the sample base posses 100 elements of each number. The experiment was conducted by presenting each sample to the Bayesian surprise model and calculating its surprise while updating the base at the same time with the presented number. They were organized in order, thus allowing to see what happens when presenting a new sample to the model. By following that experiment is expected that the Bayesian surprise calculated during the number changes, after presenting all 100 samples of the same number, it will display a surprise peak that represents how novel that sample is according to what was previously known by the model.

As depicted in Chart 60, the surprise level of each change of sample type shows a peak, where it represents the surprise of that presented sample type to the Bayesian surprise model. As can be seen at the start of the sample presentation, for the binary number 0, the Bayesian surprise response was null since there was nothing on the surprise base. As the presentation of new samples goes near to 100 the first peak surges since at that point the surprise base is able to return a calculated surprise based on the previously learned samples. What happens with the surprise at sample 400 to 800 is that the surprise base is already filled up with the binary numbers from 0 to 3, thus their peaks are minimal or they just do not exist although their binary representation is different from all the previously presented number inside the base. That behavior happens because the previously learning surprise values obtain bits that are shared between all other numbers that are presented during sample presentation from 400 to 800. This way of calculating the Bayesian surprise suits well a model in which individual features are shared between samples.

If considering the Bayesian surprise model as depicted by Chart 60, the model that codes an agent's actions need to consider the relationship between actions. For example, if the action A, from an agent's possible action list, shares a characteristic with action

Chart 60: Bayesian surprise behavior for a set of 800 observations.



Source: By the Author.

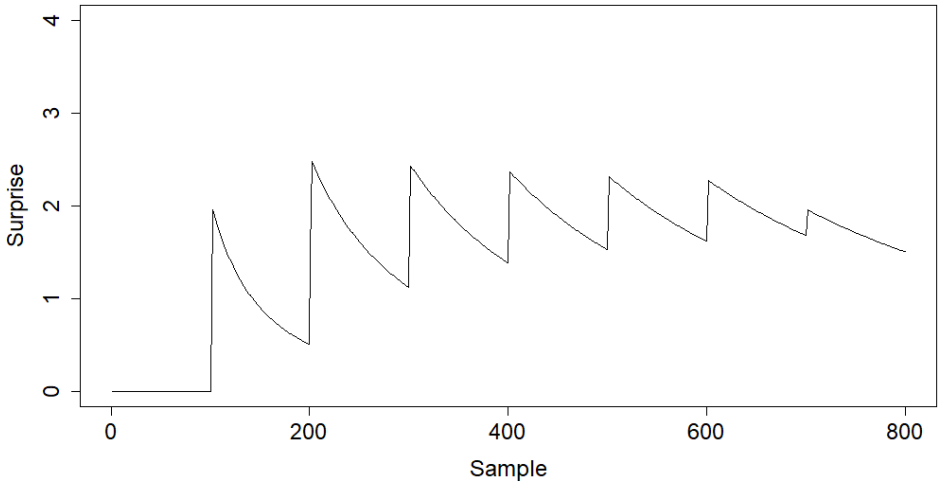
B than the Bayesian surprise will be able to calculate how similar they are and return a compatible surprise level for both. As presented in Chapter 9, all the proposed models code actions as separated individuals, where an action does not relate in any means with others. That model was coded in that way in order to facilitate how all the proposed FFSs learn and store information about what action the agent performed during the simulated POMDP. The main problem of using the Bayesian surprise is the lack of a way in discerning different actions, as proposed for Hearthstone, in a way that previously learned actions does not interfere on the calculated surprise for new ones that does not relate to them. In order to tackle that issue, the Bayesian surprise can also be calculated without considering its scaling term, shown by Equation 68,

$$\frac{N}{2\sigma_i^2} \tag{68}$$

where it represents how much the surprise will be scaled according to the sample space variation. By removing the scaling term, the Bayesian surprise behaves as depicted by Chart 61, where each peak can now be seen ranging from 0 to 2.5 and appear to be less influenced by previously learning samples. In addition, each number seems to display the same or a similar surprise peak value. Furthermore, the lowest value for each sample number type, if analyzed progressively, displays a logarithmic growth, which indicates the influence of the previously learned values on the calculation of subsequent surprise for each sample type. However, by following that model to calculate the surprise for each

HARP T-HearthBot variant, the surprise value will not be trustful yet since it will display a logarithmic growth according to previously learned samples.

Chart 61: Bayesian surprise behavior for a set of 800 observations without using a scaling term.



Source: By the Author.