HearthBot: An Autonomous Agent Based on Fuzzy ART Adaptive Neural Networks for the Digital Collectible Card Game *HearthStone*

Alysson Ribeiro da Silva and Luis Fabricio Wanderley Goes

***Abstract*—Digital collectible card games, as partially observable games based on alternating turns, such as *HearthStone*, have been the most played card games in recent years, where the main challenge is the creation of strategies capable of subdue the enemy's moves. From the artificial intelligence perspective, the space of possible strategies is large and dynamic due to the number of cards and actions combinations and also to randomness, which makes the design of efficient autonomous agents a hard problem. This paper presents *HearthBot*, an autonomous agent that plays *HearthStone* through an adaptive neural network inspired in the fuzzy adaptive resonance associative map and adaptive resonance theory map. This paper also proposes a new mechanism to categorize and predict information to overcome the overgeneralization problem from those networks. Furthermore, the proposed solution was implemented as a parallel adaptive neural network for a graphics processing unit that achieves a performance compatible with the ones obtained for deep learning methods. *HearthBot* win rate was evaluated in two experiments playing against a Monte Carlo tree search heuristic with competitive decks on a *HearthStone* simulator called *Metastone*. Results show that the proposed solution allows HearthBot to obtain an average win rate performance of 80% against known decks and 70% against unknown decks.**

***Index Terms*—Autonomous agent, adaptive resonance associative map (ARAM), adaptive resonance theory map (ARTMAP), fusion architecture for learning cognition and navigation (FALCON), fuzzy adaptive neural networks, *HearthStone*, neural networks.**

I. Introduction

DIGITAL collectible card games (DCCG) such as *Magic: The Gathering* (M:TG) and *HearthStone* have been the most played card games in recent years [1]–[3]. They are partially observable games based on alternating turns between two players, where each player has a deck of cards composed of minions and spells. These cards can be played on each turn in a battlefield to defend or attack a player [3]. The main challenge for players in DCCG is the creation of strategies capable of subdue the enemy's moves in order to reduce the opponent's health to zero and consequently win a match. However, from the artificial intelligence perspective, this space of possible strategies is large and dynamic due to the number of cards and actions combinations and also to randomness (chance), which makes the design of efficient autonomous agents a hard problem [4].

In order to tackle this problem, there are two main models on which an autonomous agent, or bot, can be built: symbolic and emergent [5]. The symbolic solutions use data structures, algorithms and heuristics, and handcrafted nonadaptive 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 nonsupervised learning, that learns without the intervention of a specialist [6].

Typically, symbolic solutions to implement autonomous agents for collectible card games, such as *M:TG*, are based on the Monte Carlo tree search (MCTS) [1] and its variants with min–max heuristic and alpha–beta pruning [4], [7]. Most of the symbolic and emergent solutions to control agents come from robotics, and they have being developed since 1970s [8]–[11] with the assistance of neural networks [4] and knowledge-based systems [12]–[15]. These networks can range from adaptive resonance theory (ART) first proposed by Carpenter and Grossberg [16] and further extended as its fuzzy counterparts [8], [17], [18] to fully brain-like structures such as the *Temporal Context Machine* created by Weng *et al.* [9] and even deep learning neural networks [19]–[21].

DCCGs are a partially observable Markov decision process (POMDP) [35], because a player does not have all the information about the game state. First person shooter (FPS) games, such as *Unreal Tournament* [6], are also POMDP, in which ART networks have been successfully used to build autonomous agents. This paper presents *HearthBot*, an autonomous agent for a DCCG called *HearthStone*, in which for a set of competitive decks, it learns an strategy to play with. The agent plays the game using an adaptive neural network that relies on fuzzy ART multichannel adaptive resonance associative map (ARAM) and adaptive resonance theory map (ARTMAP) [6] assembled in a similar way to the fusion architecture for learning cognition and navigation (FALCON) [11]. Although *HearthStone*

Manuscript received December 17, 2016; revised August 19, 2017; accepted August 20, 2017. Date of publication August 23, 2017; date of current version June 14, 2018. This work was supported by CAPES. *(Corresponding author: Alysson Ribeiro da Silva.)*

The authors are with the Department of Computer Science, Pontificia Universidade Catolica de Minas Gerais, Belo Horizonte 30535-901, Brazil (e-mail: alysson.ribeiro.silva@gmail.com; lfwgoes@yahoo.com.br).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2017.2743347

has a smaller number of cards and mechanics, the numbers of sequences of actions that can be generated in a game are comparable to the ones stated for the *M:TG* game [1], [2], thus being unfeasible to build an efficient symbolic solution. This remarks that an emergent solution could be appropriate to create an autonomous agent for *HearthStone*.

To evaluate *HearthBot* performance, experiments were conducted in a *HearthStone* simulator called *Metastone* [22]. The proposed agent plays against one preprogramed symbolic agent that uses an MCTS heuristic. *HearthBot* was trained and evaluated playing against random and MCTS agents, using recent competitive decks. Results show that *HearthBot* reaches an average win rate of 80%. The main contributions of this paper are as follows:

1) an emergent agent build on top of a fuzzy ART adaptive neural network for the game *HearthStone*;
2) a new mechanism to categorize and predict information that improves the fuzzy ART adaptive neural network in overcoming its overgeneralization problem for the proposed *HearthStone* modeling;
3) a parallel fuzzy ART adaptive neural network implemented for a graphics processing unit (GPU) that achieves a performance compatible with the ones obtained for deep learning methods.

The rest of this paper is organized as follows. In Section II, related works for agent controlling in DCCG are presented. In Section III, the card game *HearthStone* is presented. Next, in Section IV, the original fuzzy ART multichannel adaptive neural network based on ARAM and ARTMAP is introduced with some discussion on its drawbacks. Furthermore, Section V proposes an improved neural network. Next, in Section VI, *HearthBot* and its modeling with the network are presented. Section VII outlines the experimental setup, while Section VIII presents and analyzes the experimental results. This paper is concluded in Section IX.

II. Related Works in Agent Controlling for DCCG

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 [23], [24]. Ward and Cowling [23] proposes the use of MCTS to select cards to play on each turn in M:TG. Like *HearthStone*, M:TG 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 [25], 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.* [26] propose 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 and Flory [27] proposed an extension to Upper Confidence Tree and applied to a card game called Urban-Rivals.

With respect to card evaluation, Bursztein [24] 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. In a previous work [28], an agent was proposed with a search procedure to generate strategies for a set of cards to be played in a turn for *HearthStone*. This agent uses a knowledge-based system to support the construction of those strategies through an inference engine.

In [29], the authors propose an agent based on reinforcement learning to allow virtual nonplayable characters (NPCs) to behave more humanlike. The authors claim that their goal was to make the game more interesting considering the premise that a humanized NPC gives a better gaming experience to players than a robotic one. In contrast, Liu *et al.* [30] created an agent based on MCTS to allow a certain degree of adaptability inside the game, thus promoting a higher level of entertainment. In [31], the authors created agents that play Pacman with the support of neural networks to promote a better game play experience to players in social medial platforms like Facebook.

One of the most related works was the proposal of an agent controlling architecture by Tan [11], 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 allowing an agent to navigate through a hypothetical minefield. A further extension to the FALCON model was the temporal difference FALCON [32], where temporal difference methods were incorporated to allow reasoning considering time. In a recent effort to control agents in real time, Tan [11] 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*.

III. HearthStone

In the DCCG universe, *HearthStone* is one of the most played games in the last year [3], [28]. 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 30 points, and each avatar also have extra health points stored as armor points. A player also has resources called *mana* crystals and a battlefield illustrated by Fig. 1.

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 seven 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.

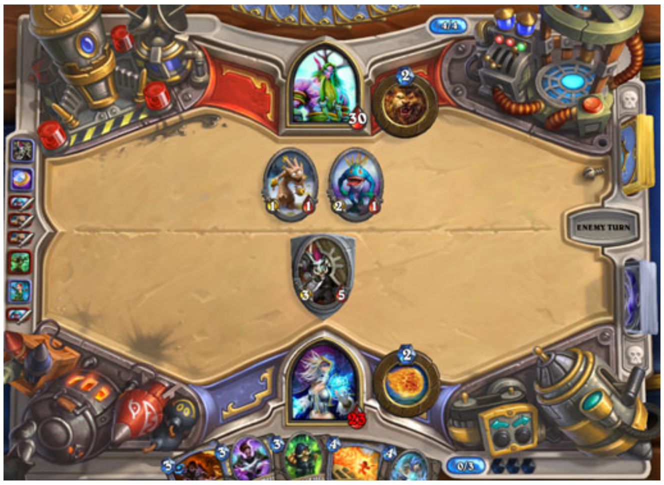

Fig. 1. *HearthStone* battlefield with the allied hero avatar on bottom and the enemy hero avatar on top. The middle represents the battlefield itself and it can contain only minions. Image obtained from a *HearthStone* online match from Blizzard's Entertainment all rights reserved [3].

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 resources (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 only *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 and use his hero power or minions to attack characters (minions or hero) and particularly combining cards, that is, playing *combos*. In this paper, we thus define a combo as a group of related cards played in the same turn, independently of order. The order in which cards are played influences the combo effectiveness and efficiency in practice, but it is a concern of the game strategy, which is not explored in this paper. The combination of card's effects is crucial on building creative decks. Depending on how cards are combined, for instance, a negative effect can become a positive one.

A. Environment

An environment 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 eight variables to allow its usage by *Hearthbot*, where $E = e_1, ..., e_8$. The variables $e1$ and $e2$ represent the summarized amount of health and armor for the player and opponent, respectively, e_3 and e_4 represent the total amount of cards in the hands 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 allows further generalization by *HearthBot* when representing more than one environment, thus allowing a concise categorization of a set of environments.

B. Cards and Actions

In *HearthStone* and in many games, an action is represented by events that happen in a time flow interacting within game components, where a time flow is a mechanism that can simulate time. 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. Events can interact with others allowing cards to influence how the game will behave, for example, if a player plays 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, especially other cards, each action can be coded as a vector composed of a source and 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 card A are encapsulated inside that card; this resembles a compact model avoid in representing all possible events that exist inside cards with semantic structures such as trees.

However, if coding only sources and targets, it is impossible to identify what kind of events happened between card A and 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 such as 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, $s_i \in [1,\ 1125]$ is the source of the action for all possible cards on the game, and $t_i \in [1,\ 1125]$ is 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].

C. Strategy as a POMDP Path

A POMDP [35] is described as a tuple of the form $(S, A, T, R, \Omega, O, \gamma)$, where S is a set of states, A is a set of actions, T is 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 outcomes from performing actions, Ω is a set of observations, O is a conditional observation probability, and γ is a discount factor. The main utility of this process is in coding

HearthStone environments, or what a player is seeing, into states used by a search procedure, thus allowing an agent in knowing what he already done. At each state, a player needs to decide, selecting actions to perform, what he will do in order to win a game. This collection of decisions over time is defined in this paper as a strategy, where, on each state, an agent has to select an action to be performed considering that it will receive an outcome in the form of a reward. A complete strategy is composed of all states reached and actions used to reach those states in a determined order, thus allowing to maximize an overall reward over time.

D. HearthStone Search Space Size Assumptions

In order to estimate how many POMDP states the proposed *HearthBot* needs to deal with, some assumptions were made:

1) each player can have a maximum of 30 health points;
2) only two copies of the same card in the deck.
3) cannot steal cards from enemy's deck;
4) cannot duplicate cards from their decks;
5) each card has only one instance, where cards affected by buff effects, that change their stats, will not be considered new instances;
6) a player has 100 hero cards plus 717 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 possess 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 equal to $1.78 \times 10^{23} \times health \times mana$, and for two players, it is equal 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 only the quantity of states in a POMDP and not paths that can be formed by combining them.

It is hard to calculate how many actions a player can perform per state, since it will depend on each battlefield configuration and hand cards. Actions can also repeat from environment to environment; thus, it is unfeasible to estimate how many action an agent can perform for each state. However, it is possible to estimate the total amount of actions that an agent can perform for the proposed tuple model as $4 \times 1125 \times 1125 = 5 \times 10^6$. Each turn is guided by randomness and generates a maximum visibility window of $4 \times 7 \times 10 = 280$ actions. Assuming an optimistic scenario, where each state allows us to perform an average of 50 actions, a player will need to handle $50 \times 2.85 \times 10^{51}$ transitions for the estimated amount of states calculated for two players.

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 equal to $\sum_{p=0}^{m} (2.85 \times 10^{51})^p$, where p is the size of the path and m is the maximum allowed path size. This estimation cannot be calculated easily, since $m = \infty$. If estimating only 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 [33]. 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.

E. Action Representation

In order for the action and environment models to be suitable to *Hearthbot*, either needs to be coded as feature vectors. The environment E can be coded directly with a vector having eight positions, one for each variable, but actions are not trivial to code. If the action model is coded bitwise, one position per action, like was done for *Unreal Tournament* [6], performance issues could arise due to large overhead of data used to represent a *HearthStone* game context. This stems from the fact that a vector with 2250 variables seems too large to be managed without increasing the complexity of any operations within it. The binary code can be used instead, but it also has unnecessary overhead proportional to the amount of bits used to code all the actions. To avoid this problem, in this paper, it is proposed that actions should be coded within an action spectrum that ranges from 0 to 1, where each element of the action model is coded as $1/id$. The id variable represents the number of the card that ranges from 1 to 1125 or the number of the action type ranging from 1 to 4, thus leading to a compact representation of an action.

While representing an environment and actions with feature vectors, the agent can perform its reasoning through an emergent solution with the support of an adaptive neural network such as the fuzzy ARAM or fuzzy ARTMAP. Those networks rely on ART operations, and in this paper, they were used to control *HearthBot*, since they are suitable to control agents in real time and also allowing a multidimensional pattern mapping with incomplete information from the environment E. It is important to note that those networks have problems that refer to an overgeneralization, data corruption, and low prediction reliability [6], [8], [11], [17], [18], [32], [34] that can affect the performance of *HearthBot*. To overcome those problems, in this paper, a new mechanism to categorize and predict information for an adaptive neural network, inspired in the fuzzy ARAM and fuzzy ARTMAP, is proposed on Section V to be used instead of ART operations.

IV. Adaptive Resonance Associative Maps

Adaptive neural networks, as the fuzzy ARAM and fuzzy ARTMAP, are used in a variety of applications, ranging from computer vision to agent controlling in games [6], [8]. Those networks allow an agent to adapt into various scenarios according to the its needs. This behavior can be useful if an environment that interacts with the agent is unknown or partially unknown. They are based on more than two ART modules that communicate with each other to form an associative memory [11], [18], [32], thus allowing an environment mapping. All fuzzy-ART-based systems are referenced in this paper as a multichannel fuzzy feature field system (FFS), since everyone available describes feature fields that interact within layers.

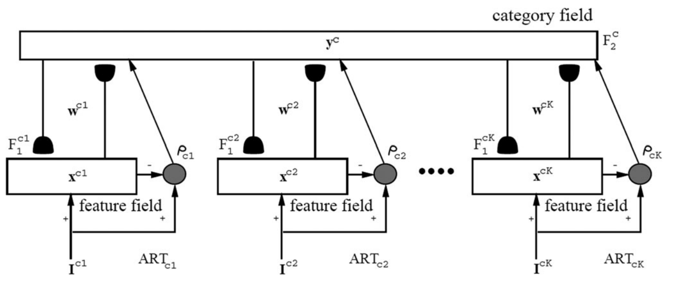

Fig. 2. ARAM architecture presented by Tan [11].

In its general form, represented by Fig. 2, the multichannel fuzzy FFS 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 $W_j \in W^{\text{all}}, j = 1, ..., t$, 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, $i \in [1, k]$ is the link to the field ci inside F_1, and k is 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 number of features coded by the field ci. All those fields also need to use the complement coding to prevent the code proliferation problem [11], [17], [18], [34]. The complement prevents the network rapid growth, but it also provides an overhead of data equal to the size of each field.

A. Network Dynamics

Its dynamics are based on a five-step routine that allows in sensing, predicting, and learning external stimulus through the presentation of an environment in each x^{ci}. The five-step routine is described as follows:

1) activation, where neurons are activated through ART 1 or ART 2 operations;
2) inhibition, responsible in selecting which activated neurons will be verified in order to achieve a prediction;
3) readout, where the selected neuron is stored in X as a prediction response;
4) resonance checking, allowing to verify the capability of a selected neuron, from inhibition, to learn the received stimulus in X;
5) learning, where the network learns the received stimulus in X by an ART 1 or ART 2 method.

The five-step routine is guided, mainly, by the parameters ρ, that acts inside resonance checking β, acting inside the learning step as a learning rate, and γ responsible for inhibiting ci fields in X.

The main problem of its dynamics is the lack of a way in representing precise information, since the fuzzy ART 1 or ART will always group received external stimulus in X, independently from the selected parameters from its five-step routine. This problem arises since the ART 1 operation calculates the similarity of the received stimulus in X as the length of X and stored weights W, and ART 2 does the same calculations based on the angle between X and W. This behavior prevents any application in representing variables precisely; thus, it is impossible to use compact models or to rely in precise responses when needed.

V. Proximity Multichannel FFS

In order to solve most of the observed problems related to the overgeneralization of the received stimulus x^{ci} coded into the neurons in the proposed *HearthStone* modeling, this paper proposes the utilization of a new metric to compute the neuron competition, resonance checking, and readout operations. Furthermore, the proposed method relies in the utilization of the resonance checking, not only for learning, but also for prediction, thus ensuring more reliable predictions, categorization, and learning.

To solve the problems presented in Section IV-A, this paper proposes the utilization of the normalized sum of the individual Euclidean distances to calculate the temperature or similarity between the stimulus x^{ci} and the neuron weight w_j^{ci}:

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

This metric can be seen as more precise than both ART I and ART II, since ART I uses 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 only 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 in [6], [8], and [18], but the overgeneralization causes data corruption after a few training steps. 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 different characteristics, that does not appear to be similar, can be falsely categorized in the same category.

A further simplification of (1) 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, (1) can be divided into two parts presented by

$$\text{norm}_d(i, j) = \frac{\sum_{z=1}^{n^{ci}} |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}} \tag{2}$$

$$t_j = \frac{\sum_{i=1}^{k} \text{norm}_d(i, j)}{m}. \tag{3}$$

as a Manhattan distance metric.

The $\text{norm}_d(i,j)$ 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} is 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 only the length is considered when categorizing. The normalization of $\text{norm}_d(i,j)$ 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.

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 multichannel fuzzy FFS, to allow its usage as a normalized metric within the proposed inhibition method described in Section V-A. 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 cannot occur, as in the common multichannel fuzzy FFS, 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, (2) 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 (2), is presented as

$$m_j^{ci} = 1 - \text{norm}_d(i,j) > \rho^{ci}. \tag{4}$$

The norm_d was subtracted from 1 to allow the ρ^{ci} to represent the percentage of matching needed in the field c^{ci} to get a resonance as in the original multichannel fuzzy FFS. It is also important to note that the calculation of the new inequality is done at the same moment when calculating (2), thus saving a lot of processing time. If the results of the inequality in (4) return true, then m_j^{ci} is set to true. If resonance occurs when $m_j^{ci} = \text{true}$ 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 Section V-A.

A. 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 paper, the inhibitory process is accomplished by getting the max neuron activity $a_j \in A$, as presented by

$$a_J = \max\{a_j : \text{for all } a_j \in A\}. \tag{5}$$

Equation (5) 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 only 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 neither as in the original multichannel fuzzy FFS, because selecting only by the similarity does not guarantee that 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, which can now be extracted directly with the max operation from the newly computed vector A. If a_j is a noncommitted neuron, then $a_j = \lambda$, where $1 < \lambda < y_j$ to ensure the selection of a noncommitted neuron if none of the $y_j \in Y$ resonate. It is important to note that a_j must be equal to λ only if the network is operating in the learning mode.

B. Prediction and Learning

A stimulus prediction for an arbitrary field ci can be obtained by multiplying (2) and (4) by $\gamma^{ci} = 0$, where $\gamma \in [0, 1]$, thus generating the following equations:

$$\text{norm}_d(i,j) = \gamma^{ci} \frac{\sum_{z=1}^{n^{ci}} |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}} \tag{6}$$

$$m_j^{ci} = (1 - \text{norm}_d(i,j)) > \gamma^{ci}\rho^{ci}. \tag{7}$$

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 $g_z \in G$, allowing the fine control over each variable influence inside the cognitive code matching. This form of coding the neuron similarities is represented by

$$\text{norm}_d(i,j) = \gamma^{ci} \frac{\sum_{z=1}^{n^{ci}} g_z |x_z^{ci} - w_{jz}^{ci}|}{n^{ci}}. \tag{8}$$

For a prediction to be achieved, the full cognitive code matching and inhibition method must be performed. The cognitive code matching could follow either (6) for common prediction or (8) for a fine-tuning prediction and the resonance vector Y must be calculated through (7). A prediction is accomplished after performing a readout, of the selected neuron J, through the ART I or ART II operations. The full process terminates when learning, if the systems is in the learning mode, where the received stimulus is learned by using the ART II learning method from the fuzzy FFS to preserve its integrity.

VI. HearthBot

An agent needs to do three basic things as a deliberative program: sense, think, and act. From the *HearthStone* perspective, the agent needs extract information from the *battlefield*, as explained in Section III. Next, it needs to think through a reasoning process and select an action to be performed during its turn. This section describes the architecture of *HearthBot* that enables it to do this three-step process.

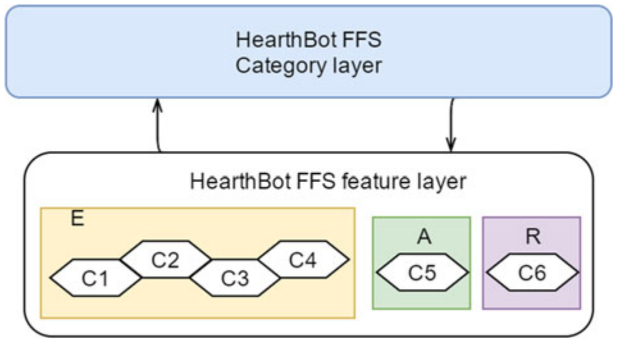

Fig. 3. *HearthBot* architecture within the proposed FFS, where the fields c^1 to c^4 represents the environment, c^5 the action, and c^6 the received reward for executing the action in c^5.

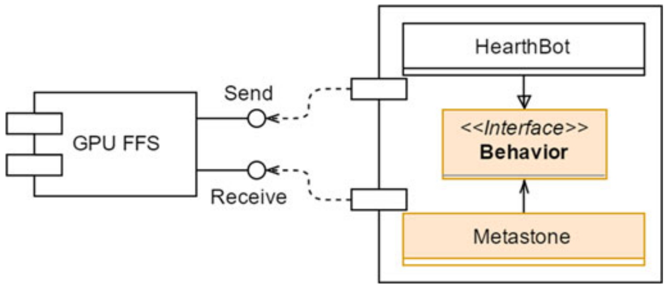

Fig. 4. Component diagram of *HearthBot* as a *Metastone Behavior* interface. The communication scheme between the improved fuzzy multichannel FFS inside the GPU and *Metastone* simulator as a separated process occurs through the interfaces *Send*, send a prediction request, and *Receive*, receive the prediction as an output.

```
input : Environment $E$, PossibleActions $A$, FFS
        $reasoner$, LocalReasoner $lr$, Boolean $learning$
output: N/A
begin
    Action $act$ = NULL
    if ($learning$) then
        $act$ = SelectAction($A$)
        Reward $r^{old}$ = GetReward($E$)
        Reward $r^{new}$ = $lr$.SimulateAction($act$)
        Reward $r^{real}$ = $r^{new} - r^{old}$
        $reasoner$.Send($E$, $act$, $r^{real}$, $learning$)
    end
    else
        Rewards $R = \emptyset$
        foreach $a_i \in A$ do
            $reasoner$.Send($E$, $a_i$, $r^{real}$, $learning$)
            Reward $r$ = $reasoner$.Receive()
            $R$.add($r$)
        end
        $act$ = max($A$, $R$)
    end
    Return $act$
end
```

Fig. 5. *HearthBot* as a *Metastone Behavior* interface. The input parameter e represents the perceived environment from a *HearthStone* game, a represent all the possible actions that the agent can execute at a turn, $reasoner$ as the proposed multichannel FFS, lr as a local reasoner to simulate the executed action, and $learning$ as a boolean to identify if the network is in the learning mode.

A. FFS Architecture for HearthBot

The architecture of *HearthBot* within the proposed FFS is depicted by Fig. 3, where the environment E was decomposed into four subfields c^1 to c^4, the action field represented by c^5, and a reward field c^6. When decomposing the environment into more than one field, it is 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. The fields are $c^1 = \{e^1, e^2\}$, $c^2 = \{e^3, e^4\}$, $c^3 = \{e^5, e^6\}$, and $c^4 = \{e^7, e^8\}$, where c^1 represents health, c^2 the total amount of cards in hand, c^3 the total amount of minions on a battlefield, and c^4 the minion score for all minion on a battlefield. All the fields were coded with the model introduced in Section III. The field c^5 was used to represent the action that *HearthBot* performed when observing the environment E. Finally, c^6 was used to store the reward received from the action performed on the observed environment E.

1) HearthBot as a Metastone Interface: HearthBot architecture is depicted by Fig. 4, where it acts as a *Metastone* interface that communicates with the proposed FFS, inside the GPU, through the *Send* and *Receive* interfaces. The *Send* interface is used to request the processing of the environment E, Action and Reward coded by the fields c^1 to c^6, and the *Receive* gives a response to *HearthBot* as a prediction following the process described in Section V. This received prediction is then used by *HearthBot* to make a decision of what action to perform next for all turns in a game until someone gets defeated.

2) HearthBot Algorithm: HearthBot agent three-step routines are performed through the *Behavior* interface, as illustrated by Fig. 4. Its algorithm, in Fig. 5, relies in sensing the *HearthStone environment* and coding it through an environment, action, and reward coding mechanisms introduced in Section III. After coding all the variables, a message string is built for sending it to the FFS input variables at I, as specified in Section V, to be normalized and copied to X to use the prediction or learning interfaces. After sending the message, *HearthBot* can obtain a response string using the *Receive* interface from the FFS on the GPU if the previous sending message was in prediction mode.

When in the learning mode, the algorithm selects an action a_i from an action selection policy through the function $SelectAction$; this action is then used to calculate the initial reward or 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 paper 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 allows us to simulate actions without changing an environment E. After the new reward calculation in r^{new}, the real one can be calculated by $r^{\text{real}} = r^{\text{new}} - r^{\text{old}}$. When $r^{\text{real}} > 0$, then there was a positive change in favor to the agent; otherwise, the change was negative.

If not in the 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 [11] 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 and 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 important 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 a score $r \in [-100, 100]$. This score is normalized by *HearthBot* inside the GetReward() function by

$$\text{reward} = \frac{100 + \text{heuristicReward}}{200}. \tag{9}$$

VII. Experimental Setup

All experiments 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 FFS was implemented in CUDA/C++ and executed in a Tesla K20 that has 2496 cores and 5 GB of memory. The total amount of memory that the Tesla K20 has also limited the total amount of neurons that the proposed FFS can handle, up 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 [22]. It encompasses the last season updates to this date from *HearthStone*. All the simulator routines were implemented with Java in [22]; due to this fact, the proposed FFS was programed to communicate with Metastone through the Java Native Interface, thus avoiding bottlenecks from interprocess communication.

A. *Experimental Design*

In this paper, *HearthBot* is compared with an MCTS agent available in the *Metastone* simulator. Due to the high number of possible decks, approximately C(1125, 30) $= 8.7 \times 10^{67}$, it was decided to use a single deck per hero avatar, which totals nine decks. 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 FFS learning and generalization capacity. Before the evaluation of *HearthBot*, this paper also provides an evaluation of the MTCS from *Metastone* against a random approach to guarantee that its behavior was not guided fully by randomness. In 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. In the second set of experiments, called *exploiting*, *HearthBot* was trained with the same decks, but instead is tested with nine different unobserved decks. This allows us to verify if *HearthBot* generalizes against new decks. Moreover, the cognitive code growth and analysis was also made, in Section VIII, during the *training* experiment, thus allowing to observe the internal behavior of the proposed FFS.

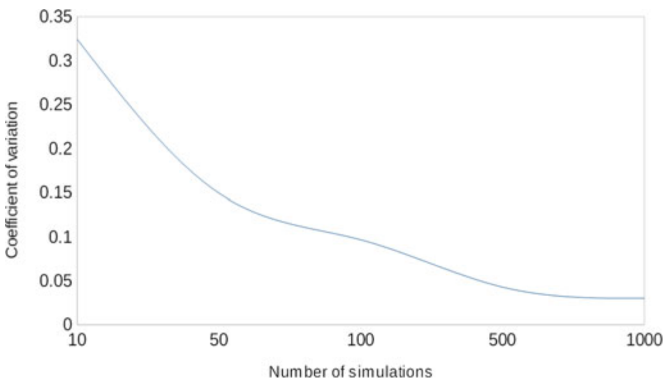

Fig. 6. Coefficient of variation represented by the Y-axis of simulations, varying from 10 to 1000.

1) Parameter Choice: In the literature, a multichannel fuzzy FFS is typically used with a high resonance criterion, above 80%, to allow some degree of specialization. For HearthBot, three configurations were used; they are specified in Table II. The configuration *reso-80* has 80% resonance in order to allow some specialization, in contrast, configuration *reso-90* was created to allow a neuron matching above 90%, and finally, the configuration *reso-90+* was used to allow near precise matching with a vigilance criterion above 95%. The last variable of each parameter set was equal to 0 for the reason that the reward field is shared between cognitive codes from the same category, thus allowing a concise categorization. Finally, all the learning rates were configured equally to the resonance of each configuration to allow fast learning.

2) Simulations: To select an adequate quantity of simulations per deck to be performed during the *training* and *exploiting* experiments, a coefficient of variation, presented by Fig. 6, 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 equal to 10, 50, 100, 500, and 1000 with a total of 166 000 simulations. The total amount of simulation per deck was limited to 1000, since the experiment has an unfeasible computational complexity.

As presented in Fig. 6, the coefficient of variation for each number of simulations ranges from 0.35 when running ten simulations to a near stable 0.05 when running 1000 simulations. Despite a reliable range equal to [0.15, 0.1] for 50–100 simulations, the usage of 1000 simulations per experiment to obtain a more precise win rate response from *Metastone* was preferred.

During the *learning* experiment, *HearthBot* training was accomplished by playing with each deck against MTCS for all the decks presented in Table I. This training stage constituted 1800 simulations per deck and a total of 16 200 simulations

TABLE I
DECK CHOICES AND THE RESPECTIVE PLAY STYLE FOR *Training* AND *Exploiting* EXPERIMENTS

	Training		Exploiting	
Hero	Deck name	Play style	Deck name	Play style
Warrior	Control Warrior	Defensive	Tempo Warrior	Hybrid
Warlock	Zoolock	Aggressive	Handlock	Aggressive
Druid	Midrange Druid	Hybrid	Egg Druid	Defensive
Priest	Control Priest	Defensive	Dragon Priest	Hybrid
Rogue	Malygos Rogue	Aggressive	Oil Rogue	Hybrid
Mage	Tempo Mage	Hybrid	Concede Mage	Hybrid
Shaman	Aggro Shaman	Aggressive	BloodLust Shaman	Aggressive
Hunter	Face Hunter	Hybrid	Midrange Hunter	Hybrid
Paladin	Secret Paladin	Defensive	Aggro Paladin	Aggressive

TABLE II
PARAMETERS CHOICES FOR EACH TEST FROM *Training* AND *Exploiting* EXPERIMENTS

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

for the full training. This experiment was performed for all the three configurations in Table II, which gives a total of 48 600 simulations for each deck. When performing, in *learning* experiment, *HearthBot* was evaluated through 1000 games against each trained deck resulting in a total of 243 000 simulations considering all the three configurations in Table II.

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 I. The win rate performance was extracted as in the *training* evaluation, with 1000 simulations each in a total of 81 000 simulations. This experiment was also performed using the configuration *reso-90* to allow 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 amount of games evaluated. All the win rate performance was finally evaluated with the geometric mean for all the experiments, thus avoiding overevaluation from different decks for each experiment.

VIII. RESULTS

The proposed FFS displays a neuron usage, calculated as the total amount of neurons used to codify all the training data, depicted by Fig. 7, proportional to the precision described by the three configurations *reso-80*, *reso-90*, and *reso-90+*. For instance, the lowest values of usage, below 0.5%, were from the Druid, Mage, Shaman, Hunter, and Paladin from *reso-80*. This stems from the fact that those decks have cards considered

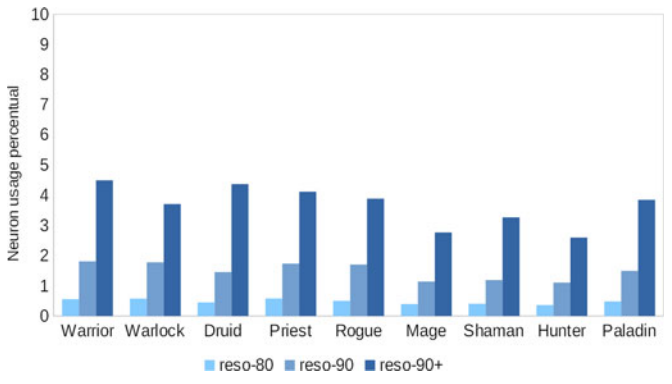

Fig. 7. Neurons usage represented by the Y-axis as the total amount of the FFS capacity was used by the configurations A, B, and C for each trained deck for each hero avatar.

similar by the FFS, thus allowing a much 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 only that the effects have a higher resonance criterion, 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 FFS had a considerable lower memory capacity, it could be practically impossible to use it.

Neuron growth can be a problem if it happens too fast because it could easily get beyond the FFS limits; for *HearthBot*, it happens in a quasi linear manner, as depicted in Fig. 8. For the configuration *reso-80*, *HearthBot* uses lower

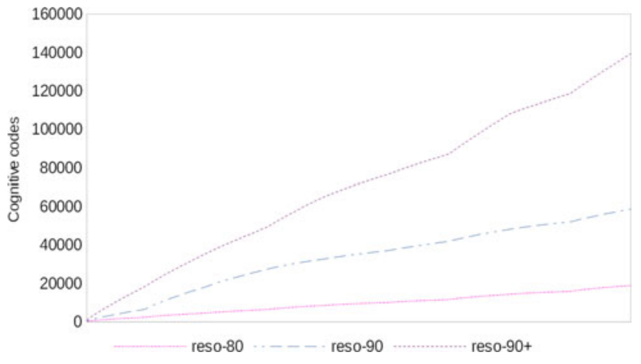

Fig. 8. Neuron growth geometric tendency for each trained deck for each configuration A, B, and C, where the Y-axis represents the total amount of neurons used by the FFS and the X-axis represents the total amount of training, from 0 to 1800, for each deck.

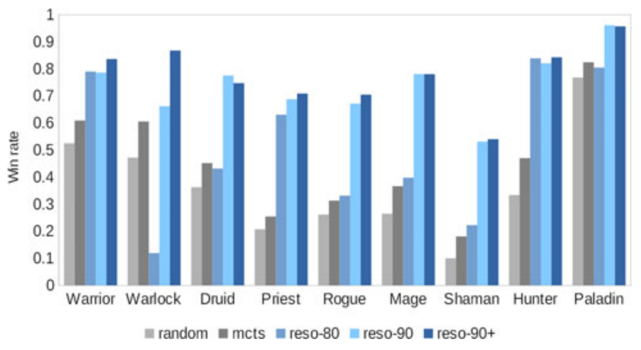

Fig. 9. Random, MCTS, and HearthBot win rate geometric tendency of the performance against all the decks of the experiments from the *training* experiment.

quantities, below 20 000; this happens due to the lower resonance criteria. In contrast, the configuration *reso-90* uses lower values at the beginning and higher values above 20 000 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. This 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, which exceeds 100 000 neurons when reaches 70% of the training stage. This growth behavior is also explained by the resonance criteria for each configuration, which has a tradeoff between more precision and code size or more generalization and less precision in environment representation. This result shows that the proposed architecture could be used with a commercial GPU without memory limitations instead of ones manufactured to scientific domain as the Tesla K20.

A. MCTS Performance Analysis

The performance of MCTS from *Metastone*, in Fig. 9, was evaluated against the MCTS and a random approach. When playing against the random approach, the MCTS performs quite well for almost every deck. The lowest win rates obtained were against the Secret Paladin deck, ranging from 0.01 playing with the Aggro Shaman to 0.19 with the Zoolock. A low win rate, below 0.3, can also be found against other heroes like the Tempo Mage versus Control Warrior or Control Priest versus Midrange

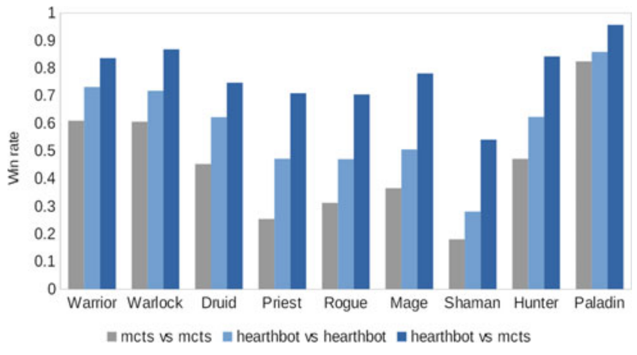

Fig. 10. Random, MCTS, and HearthBot win rate performance against all the decks of the experiments from the *training* experiment.

Druid, but the paladin deck was the strongest enemy. This behavior against the Secret Paladin could be explained as an advantage of the deck against all others considering that both, MCTS and random, use randomized approaches. In contrast, the Aggro Shaman was the weakest enemy, losing at least 70% of the time; this behavior is similar to the Secret Paladin.

When playing against itself, MCTS versus MCTS and *HearthBot* versus *HearthBot*, the expected behavior was that the win rates should be close to 50%, but it did not happen. For all the decks, as depicted by Fig. 10, 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 win rates ranging from 72% to a maximum of 95%. Furthermore, all the win rates from MCTS are slightly better, around 10%, to play with all the decks than the random approach.

B. 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 compared with all others. For example, in Fig. 9, the average win rate for the Paladin deck is higher than everyone else. This behavior is recurrent for each hero, where each has its own average win rate signature, and it occurs because each deck has a specific set of cards allowing players to perform actions that in general subdue part of an enemy's moves. For instance, the Paladin deck is composed of many secrets; 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 not tied to *HearthBot* capacity in subduing moves, but rather on the essence of a deck, since an 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.

C. Overall Performance and Discussion

HearthBot overall performance during *training* experiment is shown in Fig. 9. This graph represents the geometric tendency from the raw performance, and it was obtained during *training* experiment. As depicted in Fig. 9, 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.

The results presented in Fig. 9 also show that *HearthBot* overcomes both the 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 tendency, 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 shown in Fig. 9. 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, thus allowing 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 has 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 below from its counterpart in configuration *reso-90*, which 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 amount of neurons used.

D. Overall Performance Against Unobserved Decks

Overall, as shown in Fig. 11, the geometric tendency was 21% greater in favor to *HearthBot* compared to MCTS. The worst tendency 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 FFS 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 only 48%, and this means that it loses more than wins against the unknown opponents; nevertheless, this was the only win rate performance below 50%.

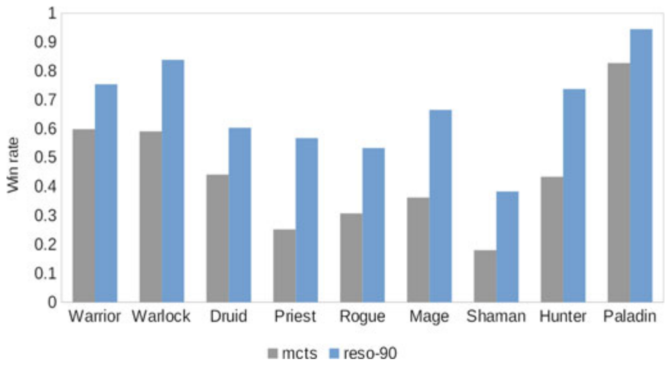

Fig. 11. MCTS and HearthBot win rate geometric tendency when performing against all the decks from the *exploiting* experiment.

HearthBot performed better than the MCTS, but it is clear that the difference 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 one obtained on the *learning* experiment, concluding that *HearthBot* seems better against the MCTS for trained and unobserved decks.

IX. Conclusion

This paper introduced *HearthBot*, an adaptive agent that relies on an improved adaptive neural network based on ARAM architecture employing a FALCON in a reactive manner. The proposed agent was controlled inside a *HearthStone* simulator called *Metastone* as a *behavior* interface, where its performance was evaluated playing against competitive decks. The agent learns how to play against competitive decks and also used the learned knowledge to play against unknown decks.

When performing against the MCTS, *HearthBot* obtained an average win rate performance near 80%; thus, it completely overcomes the MCTS from *Metastone*. It was observed that this win rate was tied to the quality of the decks used, which indicates that each deck has clearly advantages over each other deck. The randomness of the simulator was also a crucial aspect, considering that it contributes to a high coefficient of variation when calculating win rates. In overall comparison to the MCTS performance, *HearthBot* performed approximately 30% better. Despite its performance against known opponents, *HearthBot* performed as well for the unknown ones. It scored an average win rate performance near 70%, which indicates that it can abstract learned information on unknown environments.

With respect to the proposed FFS, its performance inside a dedicated GPU was clearly perceived when processing large amounts of cognitive codes. In comparison to approaches that can handle neurons amounts near 2000–5000 [6], [11], [32], the proposed FFS handles amounts above millions. This was a decisive aspect that allows *HearthBot* to overcome easily the MCTS approach, considering that the amount of neurons guides the capability of the FFS to achieve precise categorizations, what impacts directly on the quality of the predictions given to *HearthBot* when playing the game.

Despite the fact that *HearthBot* can overcome an existing MCTS inside *Metastone*, it can be improved in many points. For instance, it is also impossible to guarantee that any received stimulus from the proposed FFS gets categorized correctly with a proximity metric, at least what can be guaranteed is that the categories will always be balanced considering a resonance criterion; this is also true for the original ART I and ART II methods. Also, the proposed solution does not treat time as done with a temporal learning method like the temporal difference learning; thus, *HearthBot* was not performing at its best, leaving all those problems as future research topics.

REFERENCES

[1] C. D. Ward and P. I. Cowling, "Monte carlo search applied to card selection in Magic: The gathering," in *Proc. IEEE Symp. Comput. Intell. Games*, Sep. 2009, pp. 9–16.

[2] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in Monte Carlo tree search for the imperfect information card game magic: The gathering," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 4, pp. 241–257, Dec. 2012.

[3] Blizzard Entertainment, 2016. [Online]. Available: https://us.battle.net/hearthStone/en/

[4] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 4, pp. 317–335, Dec. 2015.

[5] J. Weng, "Symbolic models and emergent models: A review," *IEEE Trans. Auton. Mental Develop.*, vol. 4, no. 1, pp. 29–53, Mar. 2012.

[6] D. Wang and A. H. Tan, "Creating autonomous adaptive agents in a real-time first-person shooter computer game," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 2, pp. 123–138, Jun. 2015.

[7] H. Baier and M. H. M. Winands, "MCTS-minimax hybrids," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 2, pp. 167–179, Jun. 2015.

[8] G. A. Carpenter, S. Grossberg, and J. Reynolds, "ARTMAP: A self-organizing neural network architecture for fast supervised learning and pattern recognition," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 1991, vol. 1, pp. 863–868.

[9] J. Weng, M. Luciw, and Q. Zhang, "Brain-like emergent temporal processing: Emergent open states," *IEEE Trans. Auton. Mental Develop.*, vol. 5, no. 2, pp. 89–116, Jun. 2013.

[10] H. Celikkanat, G. Orhan, and S. Kalkan, "A probabilistic concept web on a humanoid robot," *IEEE Trans. Auton. Mental Develop.*, vol. 7, no. 2, pp. 92–106, Jun. 2015.

[11] A.-H. Tan, "Falcon: A fusion architecture for learning, cognition, and navigation," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul. 2004, vol. 4, pp. 3297–3302.

[12] P. Basile, M. de Gemmis, P. Lops, and G. Semeraro, "Solving a complex language game by using knowledge-based word associations discovery," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 1, pp. 13–26, Mar. 2016.

[13] C. Matuszek, J. Cabral, M. Witbrock, and J. Deoliveira, "An introduction to the syntax and content of CYC," in *Proc. AAAI Spring Symp. Formalizing Compiling Background Knowl. Appl. Knowl. Represent. Question Answering*, 2006, pp. 44–49.

[14] A. Saffiotti *et al.*, "The PEIS-ecology project: Vision and results," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2008, pp. 2329–2335.

[15] M. Tenorth and M. Beetz, "KNOWROB: Knowledge processing for autonomous personal robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2009, pp. 4261–4266.

[16] G. A. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, no. 3, pp. 77–88, Mar. 1988. [Online]. Available: http://dx.doi.org/10.1109/2.33

[17] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Netw.*, vol. 4, no. 6, pp. 759–771, Nov. 1991. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(91)90056-B

[18] A.-H. Tan, "Adaptive resonance associative map," *Neural Netw.*, vol. 8, no. 3, pp. 437–446, 1995.

[19] Z. C. Lipton and C. Elkan, "Playing the imitation game with deep learning," *IEEE Spectrum*, vol. 53, no. 2, pp. 40–45, Feb. 2016.

[20] N. Sugimoto *et al.*, "Trax solver on Zynq with deep q-network," in *Proc. Int. Conf. Field Programmable Technol.*, Dec. 2015, pp. 272–275.

[21] M. V. D. Steeg, M. M. Drugan, and M. Wiering, "Temporal difference learning for the game Tic-Tac-Toe 3D: Applying structure to neural networks," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Dec. 2015, pp. 564–570.

[22] Demilich, 2016. [Online]. Available: http://www.demilich.net/metastone/

[23] C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in magic: The gathering," in *Proc. 5th Int. Conf. Comput. Intell. Games*, 2009, pp. 9–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1719293.1719306

[24] E. Bursztein, "How to appraise hearthStone card values," 2014. [Online]. Available: https://www.elie.net/blog/hearthStone/

[25] C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in Magic: The gathering," in *Proc. IEEE Symp. Comput. Intell. Games.*, 2009, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/cig.2009.5286501

[26] N. Sephton, P. I. Cowling, E. Powley, and N. H. Slaven, "Heuristic move pruning in Monte Carlo tree search for the strategic card game lords of war," in *Proc. IEEE Conf. Comput. Intell. Games*, Aug. 2014, pp. 1–7.

[27] O. Teytaud and S. Flory, *Upper Confidence Trees with Short Term Partial Information*. Berlin, Germany: Springer-Verlag, 2011, pp. 153–162.

[28] L. F. W. Goes *et al.*, "HoningStone: Building creative combos with honing theory for a digital card game," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 2, pp. 204–209, Jun. 2017.

[29] Y. P. Fang and I. H. Ting, "Applying reinforcement learning for game ai in a tank-battle game," in *Proc. 4th Int. Conf. Innovative Comput., Inf. Control*, Dec. 2009, pp. 1031–1034.

[30] X. Liu, Y. Li, S. He, Y. Fu, J. Yang, D. Ji, and Y. Chen, "To create intelligent adaptive game opponent by using Monte-Carlo for the game of Pac-Man," in *Proc. 5th Int. Conf. Natural Comput.*, Aug. 2009, vol. 5, pp. 598–602.

[31] M. M. Hasan and J. Z. H. Khondker, "Implementing artificially intelligent ghosts to play MS. Pac-Man game by using neural network at social media platform," in *Proc. 2nd Int. Conf. Adv. Electr. Eng.*, Dec. 2013, pp. 353–358.

[32] A. Hwee Tan, N. Lu, and D. Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 230–244, Feb. 2008.

[33] W. Poundstone, *The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge*. North Chelmsford, MA, USA: Courier Corporation, 2013.

[34] A. H. Tan, "Adaptive resonance associative map: A hierarchical art system for fast stable associative learning," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 1992, vol. 1, pp. 860–865.

[35] Sondik, Edward J., "The optimal control of partially observable markov processes over the infinite horizon: discounted costs," *Oper. Res.*, vol. 26, no. 2, pp. 282–304, Apr. 1978.

Alysson Ribeiro da Silva received the Graduate degree in computer science in 2014 from the Pontifical Catholic University of Minas Gerais, Belo Horizonte, Brazil, where he is currently working toward the M.S. degree in the Electrical Engineering Graduate Program (EEGP).

He is also a student at the Computational Creativity and Parallel Computing Laboratory, EEGP. His research interests include humanoid robots, artificial intelligence, computational creativity, and games.

Luis Fabricio Wanderley Goes received the B.Sc. degree in computer science and the M.Sc. degree in electrical engineering from the Pontifical Catholic University of Minas Gerais, Belo Horizonte, Brazil, in 2002 and 2004, respectively, and the Ph.D. degree in parallel programming, parallel skeletons, transactional memory, performance tuning from the University of Edinburgh, Edinburgh, U.K., in 2012.

He is currently an Associate Professor with Pontifical Catholic University of Minas Gerais. His research interests include parallel programming computing, computational creativity, and games.