Development of an Autonomous Agent based on Reinforcement Learning for a Digital Fighting Game

João Ribeiro Bezerra
Department of Computer Science
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brazil
joao.bezerra@sga.pucminas.br

Luis Fabrício Wanderley Góes
Department of Computer Science
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brazil
lfwgoes@yahoo.com.br

Alysson Ribeiro da Silva
Department of Computer Science
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brazil
william.santos@sga.pucminas.br

***Abstract*—In this work, an autonomous agent based on reinforcement learning is implemented in a digital fighting game. The implemented agent uses *Fusion Architecture for Learning, COgnition, and Navigation* (FALCON) and *Associative Resonance Map* (ARAM) neural networks. The experimental results show that the autonomous agent is able to develop game strategies using the experience acquired in the matches, and achieves a winning rate of up to 90% against an agent with fixed behavior.**

***Index Terms*—fighting game AI, game AI, reinforcement learning, neural networks**

I. Introduction

Electronic games are a form of entertainment that has been gaining more and more consumers, especially with the increasing ease of access to electronic devices worldwide. With the introduction of the possibility to play with others over the internet, competitive games have gained more space in the market [1].

Currently, the behavior of computer-controlled agents in commercial games is usually fixed, with scripted pre-programmed strategies. This makes playing against these agents increasingly distant from the experience of playing against a human opponent. For competitive players, this fixed behavior causes them to acquire bad playing habits when training against the machine. In addition, as the machine starts to have predictable reactions, the gameplay becomes repetitive and predictable [2]. The developer, responsible for creating these scripts, needs to anticipate about every kind of situation possible in the game environment, which requires effort [3].

The development of this work is motivated by the possibility of implementation of an artificial intelligence technique that makes the machine capable of learning and developing game strategies with certain autonomy. In this way, it can adapt to the way players play the game. This machine behavior has a positive impact on the game's gameplay by iterating over players' input — which is constantly changing — to offer new tactics and new challenges for the player to overcome and improve their skills [4].

Identify applicable funding agency here. If none, delete this.

The objective of this work is to develop an autonomous agent based on reinforcement learning for a digital fighting game. It is implemented using *Fusion Architecture for Learning, COgnition, and Navigation*(FALCON) and neural networks *Associative Resonance Map*(ARAM). The experimental results show that the autonomous agent is able to develop game strategies using the experience acquired in the matches, and achieves a winning rate of up to 90% against a fixed behavior agent.

In section 2, work related to the proposal of the present study is discussed. Section 3 presents the theoretical framework used. In section 4, the game used in this work, *Fighters Arena* is presented. In section 5, the implementation of the proposed agent is detailed. In section 6, the experimental methodology is presented. In section 7, the experimental results are presented. Section 8 presents the conclusion.

II. Related work

In this section, related work from the technical literature on the topics involved in this work are presented: machine learning and its use in digital games.

In the work of P. G. Patel, N. Carver, and S. Rahimi [5], the authors discuss the implementation of artificial intelligence techniques in commercial digital game agents and point out that generally the logic of simple routines is used. Considering that the challenges and requirements for digital game developers are similar to those of the academic artificial intelligence community, the authors develop an abstraction of the game Counter Strike with agents that simulate players using artificial intelligence based on Q-learning. With test runs performed using these agents, it was possible to note that their performance is superior to that of fixed-behavior agents, and that using Q-learning, agents were able to learn various behaviors based on the reward they get from their actions.

In the work of M. R. F. Mendonça, H. S. Bernardino, and R. F. Neto [4], the authors implement two forms of machine learning in a fighting game using Neural Networks — with and without Q-learning — for game agents to simulate a human player. The methods are assigned a reward function proposed by the authors and the results are compared with two other forms of machine learning. The results of the experiments indicate that the methods using Q-learning achieve better performance with human players when compared to other existing methods.

The authors S. Saini, P. Chung, and C. W. Dawson [1] have developed a way for an avatar to learn and replicate a player's playing style. Their method is to analyze the data and separate it as tactical and strategic data. A Naïve Bayes classifier is used to classify tactics for specific states, and a finite state based data machine to dictate when certain tactics are used. Implementation results were positive, but limitations were noted as to the number of variables that can be analyzed due to the use of a finite state machine. It is concluded that facing this avatar may seem repetitive in relation to a human player due to the indiscriminate way in which actions are chosen, regardless of errors and preferences.

In the work of G. Andrade, G. Ramalho, H. Santana, and V. Corruble [6], the authors implement an agent using artificial intelligence for a fighting game using Q-learning, and justify its use for simplicity and good results. The implementation proposed in the work uses a difficulty balancing system in which, given a situation, instead of using the best known option or a random option for learning, it uses actions that match the skill level of the player it faces.

In the work of A. Carpenter, S. Grossberg, and D. B. Rosen [7], a variation of the fuzzy Q-learning algorithm for training an intelligent agent to play *Ms. Pac Man*. In the strategy used, the agent analyzes the current situation of the game, considers variables that contribute to the situation in which he finds himself, and makes decisions based on these values.

In the work of D. Wang and A. H. Tan [8], the FALCON architecture (*Fusion Architecture for Learning COgnition and Navigation*) is used to implement agents in the game *Unreal Tournament* in real time, using fuzzy ARAM neural networks and combinatorial operations for comparison between the two methods. The implemented agents are able to develop strategies and explore the effectiveness of different combat tools in the game without any human intervention. With the knowledge absorbed by the agent, he is able to adapt to a new opponent or a map unknown to him in a short period of time.

In the work of A. R. da Silva and L. F. W. Goes [3], a FALCON network is used to implement an agent capable of playing the digital card game *Heartstone* using machine learning. The agent is trained in matches against agents using AI based on *Monte Carlo Tree Search*. The agent develops the ability to achieve an average 80% winning rate in matches against the same agents used in his training.

The work presented in this section, as well as the present study proposes, explore the use of artificial intelligence in digital games. Most of the work cited study competitive games or specifically fighting games, with the purpose of presenting the player with diversified and adaptive experiences, instead of the predictable ones widely used commercially. As in this article, the proposed work involve the development of adaptive gaming experiences. However, in the present study, a strategy closer to the work of [8] is used, in which the proposed agent performs actions that describe a behavior, and the neural network specializes in deciding which action is most appropriate given the current situation in that the agent is in the game.

III. Theoretical Framework

A. *Machine learning*

The reinforcement learning method consists of an artificial intelligence technique in which the agent learns to solve a given problem from his experiences of previous actions. As illustrated in Fig. 1, from the perceptions of external stimuli (s) of the environment, the selected actions generate signs of reward (R (s, a)), which can be positive or negative. negative depending on the results of the action performed [4] [5].

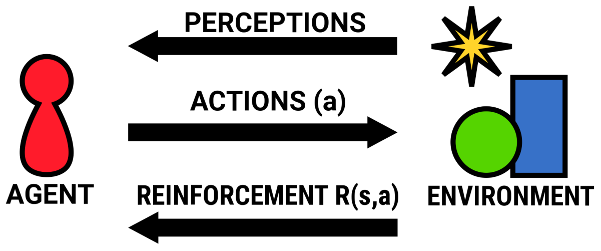

Fig. 1. Standard reinforcement learning model.

B. *Q-Learning*

Q-Learning is a reinforcement learning technique in which, based on a set of states, a set of actions and a reward function, it is possible to obtain a quality metric for each action performed in each state. States are sets of actions possible by the agent. Actions act as transitions between states, while the reward function is a quality metric of the result of the chosen action. This metric is *Q-Value*. The *Q-Values* are stored in a table called Q-Table. When making a change of state, the reward obtained changes the weight of this transition between states in the Q-Table, which, with training, becomes more accurate in predicting the result of each transition between states. The equation used to update the Q-Value is shown in (1). The γ parameter is the discount factor ($0<\gamma<1$), used to give preference to more recent rewards; The α parameter is the learning rate. The variable (t) is the current time, (s) is the external stimulus, and (a) is the selected action [4] [9].

$$Q(s_t, a_t) = (1-\alpha) \times Q(s_t, a_t) + \alpha \times (r_t + \gamma \times \max_a Q(s_{t+1}, a)) \tag{1}$$

C. *Artificial Neural Networks for Reinforcement Learning*

An Artificial Neural Network (ANN) is a model composed of interconnected elements (neurons). Each neuron receives a series of input stimuli that result in an output value, a process based on the weights associated with the input values of the stimuli. ANN works in two modes: training and execution. In training, several input values are presented for the ANN, as well as the desired output. For each situation presented, ANN modifies its internal weights to obtain the desired output. With this mechanism, the ANN finds a relationship between the input values and the output variables, and becomes able to predict the output values from a given input based on modeling the internal relationships between neurons. In execution mode, the internal weights aren't modified by the outcome [4].

1) ARAM networks: Adaptive neural networks, such as the fuzzy Associative Resonance Map (ARAM) and the fuzzy Adaptative Resonance Theory Map (ARTMAP), are used in a variety of applications, from computer vision to game agent control. These networks allow the agent to adapt to various scenarios according to their needs. This behavior can be useful if the environment that interacts with the agent is not known or is partially known. They are based on Adaptive Resonance Theory (ART) modules that communicate with each other to form an associative memory, thus allowing the mapping of the environment. ART fuzzy networks that describe feature fields that interact between layers are known as multi-channel fuzzy Feature Field System (FFS). In general, the multi-channel FFS fuzzy networks have two layers: F1 and F2, as can be seen in Fig. 2.

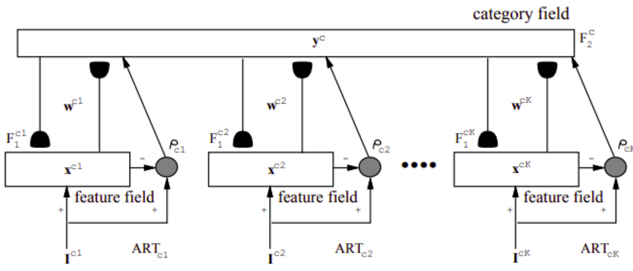

Fig. 2. ARAM architecture. Source: [10]

As seen in Fig. 2, the F1 layer (features), is where features are extracted from the environment and arranged in a coherent way. The F2 layer (categories), maintains a multidimensional pattern that has the function of connecting all fields of features of F1 in F2 using neurons. The network receives stimulus signals from the environment as input. These signals are then transformed into a feature vector, which is transferred to the elements of F1 [3] [11].

2) FALCON architecture: The FALCON architecture (Fusion Architecture for Learning, COgnition, and Navigation) is proposed in [10]. It employs a 3-channel architecture, comprising a sensory field, which represents states of the environment; a motor field, which represents the possible actions; and a *feedback* field representing reinforcement values. FALCON is a self-organized neural network based on ART, which means that it is able to evolve systematically to incorporate new information [3].

There are two main processes for using FALCON networks: retrieving knowledge and updating knowledge. When it is necessary to retrieve knowledge, we insert the vectors currently known from the input fields to retrieve the corresponding winning code. The winning code is the one with the highest activation value among all the codes that fill the network. When it is necessary to learn knowledge, we present all vectors of the three input fields and execute the knowledge retrieval process to find the winning code. The information presented is used to update the weights of the winning code. If there is no code on the network that meets the vigilance criteria, FALCON automatically creates a new code for future use. Thus, FALCON dynamically expands its network architecture in response to new entry standards in progress [8] [3] [10].

A FALCON network has 7 sets of meta-parameters. These parameters are: i) vigilance standards, $\{\rho\ 1,\ \rho\ 2,\ \rho\ 3\ \}$; ii) Learning rate parameters, $\{\beta\ 1,\ \beta\ 2,\ \beta\ 3\ \}$; iii) Contributing factors, $\{\gamma\ 1,\ \gamma\ 2,\ \gamma\ 3\ \}$; iv) Choice parameters $\{\alpha\ 1,\ \alpha\ 2,\ \alpha\ 3\ \}$; v) Learning rate parameter α; vi) Learning discount parameter γ; vii) Selection policy threshold ϵ [8] [3] [10].

IV. Fighters Arena

For this work, the competitive fighting game in development by the author called *Fighters Arena* is used. In this game, matches involving 2 to 4 players - human or machine controlled - are carried out in three-dimensional arenas with top-down view as illustrated in Fig. 3. The player can select one of several characters available. These characters share basic actions, such as walking and defending, but have different characteristics, such as: size, speed, attack animations, attack damage, etc. In a match, the players face each other in a fight in which the objective is to hit the opponent until their health bar depletes, which causes the player to be eliminated from the competition. A player is declared the winner when they are the last fighter remaining.

The game used in this work has an implementation of an avatar with fixed behavior for players controlled by the machine. In this work, it was proposed to implement a new methodology for the behavior of these players controlled by the machine using neural networks in order to make a comparative analysis of these two methodologies.

V. Agent deployment

An agent in the game *Fighthers Arena* needs to perform a three-step process to be functional: i)It must be able to perceive the game environment; ii)It must process this information to

Fig. 3. Example of a match in the game *Fighters Arena*.

analyze what action should be taken; iii)It must perform the chosen action.

A. Neural Network Architecture

For the implementation of the agent, a neural network with FALCON architecture was used. The agent chooses an opponent at random, and the game environment variables analyzed by the network are:

1) The action the opponent is currently performing.
2) If the agent's special bar is full.
3) If the opponent's special bar is full.
4) If the opponent is in a state that allows interruption into another action.
5) The remaining time of the opponent's current animation.
6) If the opponent is defending.
7) The distance to the opponent.
8) The difference between the agent's rotation and the direction the opponent is in relation to them.
9) If the agent is in a state that grants invincibility.
10) If the opponent is in a state that grants invincibility.
11) Whether or not the agent is defending.

The continuous input values are all normalized in the range [0, 1]. Logical values are 0 for false and 1 for true. Eleven basic actions were defined, as shown in Table I, which represent strategies that can be performed by the agent. One more field is used for the reward, which is calculated from the reward function at the end of each action and used in the learning process.

The reward function returns from 0.8 to 1.0 if the agent hits the opponent, being greater the more damage is inflicted; 0.6 if the opponent attacked and missed the agent or if the agent defends an attack by the opponent; and 0.1 if the agent was hit by the opponent. Any other situation results in a 0.5 reward. No reward 0 was used, as no action is completely dependent on the agent, so it was disregarded.

The agent is implemented in the game using the same structure that the fixed behavior agent and the human player use to interact with the game, making their ability to input commands and perform actions compatible with the rules that the game defines. In other words, the commands performed in the game by the network can also be carried out by a human who makes the same decisions.

TABLE I
LIST OF AGENT STATES AND THEIR RESPECTIVE BEHAVIOR

State	Behavior
jab	Approach the opponent and use a standard sequence of 3 light, fast, short-range attacks.
tilt	Approach the opponent and use a light, fast, short-range directional attack.
strong	Approach the opponent and use a strong attack that hits around the character and has a medium duration.
strongTilt	Approach the opponent and use a strong attack that is slow but has long range.
keepDistance	Move away from the opponent and use projectile moves from a safe distance.
punish	Quickly approach the opponent with a swift attack.
roll	Roll backwards, which grants intangibility from incoming attacks during the animation.
retreat	Retreat from the opponent using defensive movement options.
grab	Approach the opponent and use a grab move, effective against a defending opponent.
shield	Defend.
charge	Approach the opponent and use the ”charge” attack, which is invincible at the beginning of the animation.

B. Agent Algorithm

The three steps of the implemented agent routine use the following routine: starting from an initial state, the game environment is obtained through sensors. The algorithm then selects an action through a selection policy that, initially, encourages the exploration of possible options, partially independent of the knowledge already acquired. The results of these actions are used to calculate the reward for each action in each environment setting. As more knowledge is acquired, the network tends to choose actions that tend to result in greater reward. The chosen action is performed until the agent hits the opponent or is hit by the opponent. This event is then analyzed and the reward is calculated based on the result, thus ending a three-step routine cycle. The network interprets this sequence and updates its internal weights, generating knowledge. The routine is then repeated from the beginning.

VI. METHODOLOGY

In this section, the methodology used for the experiments of this work is presented.

A. Experiment setup

The test environment was designed so that there was a focus on the performance of the network and its learning process, limiting the number of variables and situations. To this end, the developed agent was picked to fight against a single opponent using a *script* with fixed behavior programmed using a greedy strategy. Both agents use the same character, which is designed to be the most versatile and neutral in terms of abilities, with no polarizing characteristics in their design. The arena selected for the battles is a rectangular room completely closed and without obstacles, unevenness of terrain or external elements that might interfere in the battle, to avoid interference that could alter the results in a random fashion.

All experiments were performed on an Ubuntu Linux installed on an Intel i5-6500 with 4 threads and 8 GB of memory. The code for the proposed agent was implemented in C# on the Unity engine as a component of the game, which was run at 4 times its standard speed (240 frames per second).

B. *Choice of parameters*

To carry out experiments in behavior modeling, it is necessary to configure the FALCON parameters. The parameters used are listed below.

Vigilance parameters $\{\rho1, \rho2, \rho3\}$ are defined as {0.9; 0.9; 0.8 }. Since most of the attributes in the state vector are Boolean, we set $\rho1$ to a high value to preserve a certain level of integrity of the knowledge learned. At the same time, $\rho1$ is not defined as 1.0 to allow for the compression of knowledge and to obtain a certain level of generalization. $\rho2$ was set to 0.9 for the field of action to ensure accurate combinations of Boolean variables. The reason for defining $\rho3$ as a high value for the reward field is to ensure effective learning for all similar action-state pairs. At the same time, $\rho3$ is defined as a lower value than the others in order not to completely associate the action-state pairs with the result, since it does not depend only on the agent.

The learning rate parameters $\{\beta1, \beta2, \beta3\}$ are defined as {0.2; 0.2; 0.1 }. Since the reward is immediate, $\beta1$, $\beta2$ and $\beta3$ are defined as low values for slower learning. Contribution factors $\{\gamma1, \gamma2, \gamma3\}$ are defined as {0.5; 0.5; 0.5 }, so that the most recent learning is always more relevant than the past learning, meaning the agent adapts quickly to the opponent's strategy changes.

The choice parameters $\{\alpha1, \alpha2, \alpha3\}$ are defined as {0.5; 0.5; 0.5 }. This was done to avoid any kind of invalid calculation. The learning rate parameter $\alpha = 0.5$ and the learning discount parameter $\gamma = 0.5$, which are values generally used in the literature [8]. The selection policy threshold ϵ is initially set to 1.

C. *Simulations*

In order to analyze agent performance and network learning, three groups of tests were performed. In each group, a new agent was trained following the algorithm described in Algorithm 1.

Each group was used to test the autonomous agent in matches against the fixed behavior agent, performing 50, 100 and 200 matches, respectively. The selection policy threshold ϵ was reduced linearly at the end of each match so that, in each group, in the first half of matches it was positive and the agent explored more options. In the second half of each group's matches, the agent used only the knowledge previously acquired. At the end of each game, the values of proportion of remaining life of the autonomous agent and number of neurons in the network used by the autonomous agent were counted.

Algorithm 1 Agent learning

```
Start: Initialize the network with the defined parameters and
weights with neutral value
while Game running do
    Select and perform an action using the network in accordance with the selection policy;
    if An attack by the autonomous agent or his opponent hits then
        Update the network's internal weights;
    end if
    Decrement the value of $\epsilon$
end while
```

VII. Results

The results obtained with the simulations are shown in Fig. 4, 5 and 6.

The values of the first half of the simulations matches had a greater standard deviation than the second half: 0.21, 0.19, and 0.20 for the first halves, 0.18, 0.17, and 0.19 for the second halves. This behavior was expected, considering that the agent was induced to explore more options in the first half and to act based on his previous knowledge in the second. Despite this, the second half of the matches is also not entirely consistent. This is due to the fact that, although the autonomous agent decides for actions that, from experience, would bring better results, it depends on the fixed-behavior agent to select an action that makes the result successful. Through observation of the execution of the experiments, it was possible to note that the autonomous agent was able to learn to react in a similar way to the optimum predicted for the situations in which he found himself during the matches. The autonomous agent achieved a winning rate of 90%, 84% and 89%, respectively, in the test groups. The average life remaining at the end of group matches was 37%, 28% and 37%, respectively.

Regarding the network used in the implementation of the agent, the number of neurons in the network of each test group is shown in Fig. 7, 8 and 9. Through the observation of the graphs, it is possible to notice that the progression in the number of neurons was of as expected. In all test groups, at first, the rate of growth of the number of neurons was high. As more matches were performed and situations similar to those already known to the autonomous agent were presented, this growth rate decreased until it approached zero. The number of neurons was higher in the test groups in which more matches were performed because, as a result of the greater number of matches, the agent was in contact with more different situations.

VIII. Conclusion

From the results of the experiments, it can be said that it was possible to implement the artificial intelligence technique based on reinforcement learning that made the machine capable of learning and developing game strategies with a certain autonomy. It was able to adapt to the fixed-behavior agent's strategies. The agent's performance was satisfactory in view

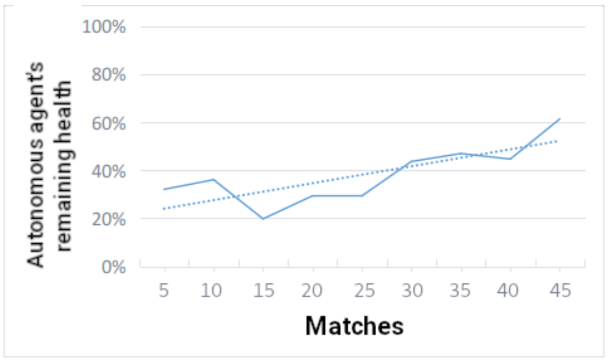

Fig. 4. Remaining agent life x Matches. Each point on the Y axis represents the average remaining life of the autonomous agent over 5 matches.

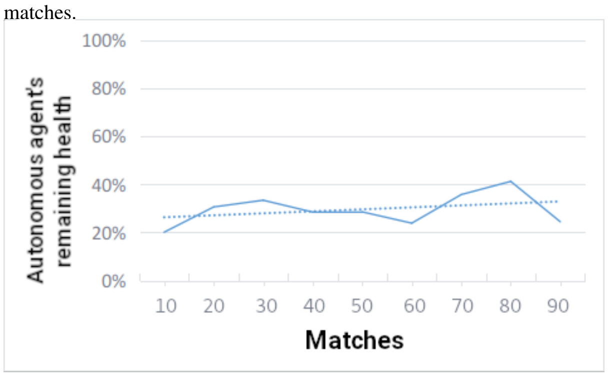

Fig. 5. Remaining agent life x Matches. Each point on the Y axis represents the average remaining life of the autonomous agent over 10 matches.

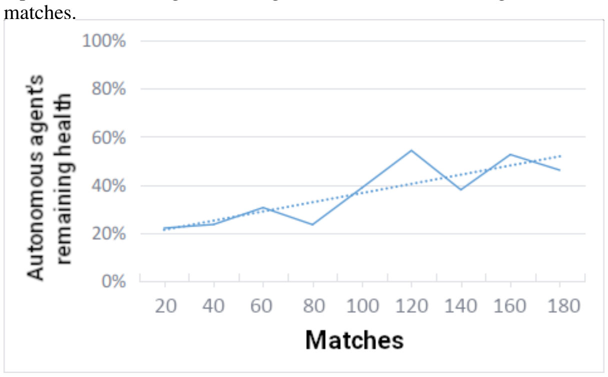

Fig. 6. Remaining agent life x Matches. Each point on the Y axis represents the average remaining life of the autonomous agent over 20 matches.

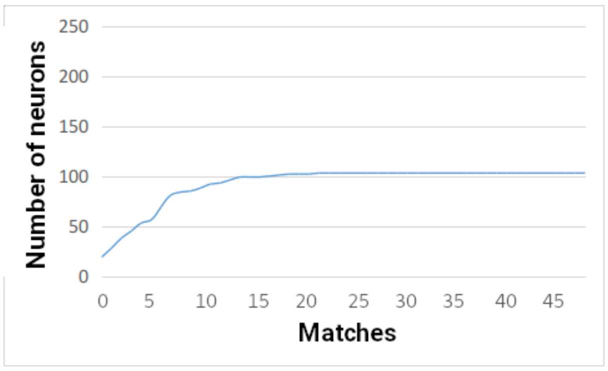

Fig. 7. Number of neurons in the network used by the agent x Matches, for 50 matches.

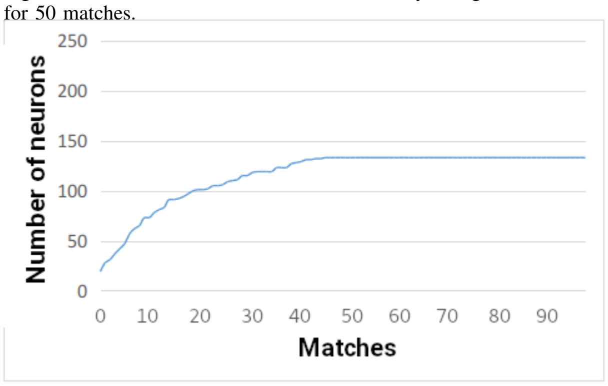

Fig. 8. Number of neurons in the network used by the agent x Matches, for 100 matches.

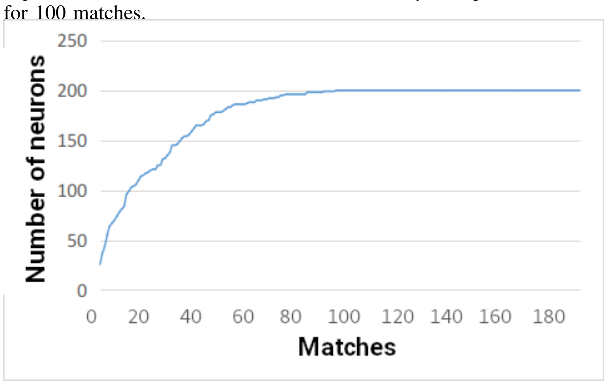

Fig. 9. Number of neurons in the network used by the agent x Matches, for 200 matches.

of the high rate of victories obtained by the autonomous agent in the experiments carried out.

As for the network used, its complexity was sufficient for there to be an efficient learning. At the same time, the network has not become complex enough for its logic to impact performance when run in conjunction with game logic, in real time.

To continue this work, complementary networks can be used for each of the states of the main network, in order to make the agent optimize the actions it performs in each of the states. This procedure can contribute to the results, for example, with the optimization of the attacks executed by the agent at a time when it selects an offensive action.

References

[1] S. Saini, P. Chung, and C. W. Dawson, "Mimicking human strategies in fighting games using a data driven finite state machine," *Information Technology and Artificial Intelligence Conference (ITAIC), 2011 6th IEEE Joint International*, 2011.

[2] T. P. Hartley and Q. H. Mehdi, "In-game tactic adaptation for interactive computer games," *Computer Games (CGAMES), 2011 16th International Conference on*, 2011.

[3] A. R. da Silva and L. F. W. Goes, "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. 1–1, 2017.

[4] M. R. F. Mendonça, H. S. Bernardino, and R. F. Neto, "Simulating human behavior in fighting games using reinforcement learning and artificial neural networks," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Nov 2015, pp. 152–159.

[5] P. G. Patel, N. Carver, and S. Rahimi, "Tuning computer gaming agents using q-learning," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2011, pp. 581–588.

[6] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: an application to game balancing," in *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Sept 2005, pp. 194–200.

[7] L. L. DeLooze and W. R. Viner, "Fuzzy q-learning in a nondeterministic environment: developing an intelligent ms. pac-man agent," in *2009 IEEE Symposium on Computational Intelligence and Games*, Sept 2009, pp. 162–169.

[8] D. Wang and A. H. Tan, "Creating autonomous adaptive agents in a real-time first-person shooter computer game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 2, pp. 123–138, June 2015.

[9] D. Pandey and P. Pandey, "Approximate q-learning: An introduction," in *2010 Second International Conference on Machine Learning and Computing*, Feb 2010, pp. 317–320.

[10] A.-H. Tan, "Falcon: a fusion architecture for learning, cognition, and navigation," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 4, July 2004, pp. 3297–3302 vol.4.

[11] 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 Networks*, vol. 4, no. 6, pp. 759 – 771, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190056B