Simulating Human Behaviour in Games using Machine Learning

Daniel de Almeida Rocha, Julio Cesar Duarte
Computer Engineering Department
Instituto Militar de Engenharia, IME
Rio de Janeiro, Brazil
{*danielrocha, duarte*}*@ime.eb.br*

***Abstract*—The study of Artificial Intelligence and, more specifically, Machine learning in games is of great interest to the gaming industry due to their wide application in several scenarios and their capabilities of simulating human behavior by non-playable characters. In online multiplayer scenarios, one of the greatest concerns is player disconnections and how to replace this player with a "good" replacement bot. In this work, we propose a machine learning based methodology to simulate the behavior of players that learns from their gaming history that generates more balance in the game. Our work uses Multi Layer Perceptron Neural Networks evaluated in a classic card game called Hearts, in order to emulate some previously defined behaviors. The results obtained in the experiments indicate that the proposed method has very good performance, since all generated models have managed to approach similar amount of victories when comparing to the behaviors that they were trained with. Through the evaluation of the results of 10,000 matches, with *gameseeds* of different matches used for training, the best result was for the model Shooter versus 3 Sheriffs getting 95.5% of the amount of wins compared to their particular bot. We also conclude that behavior learning is also clear on the difference of wins in all results depending on opponent skills, i.e., opponents who were difficult to win remain difficult to win in the simulated environment.**

***Keywords*-Machine Learning; Neural Networks; Games;**

I. Introduction

The gaming industry is a very important segment of the Brazilian economy nowadays, having made 1.5 billion dollars in 2018, 15.3% higher than the $1.3 billion billed in 2017, making it the 13th largest game market in the world [1]. Also worldwide, it made more than $134.9 billion [2]. With this growth, the concern with the quality that the games offer to the public increases, and the application of Machine Learning techniques to improve the Artificial Intelligence of these games is one of the greatest bets to reach this goal.

According to Barendregt [3], when a game is not pleasant or fun, it does not arouse interest. If it is tiring, difficult or very easy, the player may face an entertainment issue. Entertainment issues are more difficult to diagnose than technical issues, as a technical issue affects the entire user database and has tools to aid detection while entertainment problems affect only certain user segments and there is no tool to help in this type of detection.

By simulating certain behaviors, it is possible to validate strategies adopted with greater and lesser potential of victory, allowing the creation of a more interesting and attractive environment for specific users.

Microsoft has created the "Drivatar" technology initially for the game Forza Motorsport 5, which uses cloud processing to calculate the artificial intelligence of the cars against which the player runs against [4]. In this model, the opponents do not adapt to you, they are only pre-setup profiles with different skill levels. Also, artificial intelligence created by DeepMind, a company acquired by Google in 2014, was trained to play Go and defeated the best human player in the world [5]. The same artificial intelligence was also trained to play Chess and won the world's best computer program, having learned alone to play in less than four hours [6].

In a digital game where the player can interact with other players, not always those other players are real people, and, many times, are controlled by the computer itself, being called robots, or simply, bots. The intelligence of these robots is defined by a set of algorithms and heuristics that can be considered efficient for a given public but inefficient for another. When it is possible to understand a player's behavior, we can make the robot behave in a specific way if that player is absent, helping the handling of the environment so that the game becomes more interesting and attractive.

Using 2016 statistics, 33% of the Brazilian internet users who play games, play offline [7], that is, more than half of the players prefer to play online, where they depend on a good connection to the internet and other connected players. In this environment, when a player is disconnected, the whole game is affected. In the worst case scenario, it is necessary to start a new game, while, in the best case scenario, that player is replaced by a bot. Even in games that approach the best case scenario, bots usually do not behave at the same level as the player that was disconnected, unbalancing the game and generating frustration for the rest of the players. In the case of the substitution by a bot that behaves similarly to the player that was disconnected, it is possible to run the game without taking completely different paths while also increasing the possibility of the return of that player, as he will know that he has not had great disadvantage while was disconnected.

The main objective of this work is to propose a methodology to simulate the behavior of a player, using machine

learning techniques, based on their gaming history that generates more balance in the game. As a secondary objective, we intend to validate the performance of a behavior simulation in a real scenario where a database and a working environment already exist. Due to the difficulty of finding public gaming history datasets, our matches will be simulated in a controlled environment, played by different bots with several behaviors.

This article is divided into seven other sections. Section II provides the basic concepts of Games Theory and Artificial Intelligence in digital games, exemplifying machine learning techniques. Section III describes and compares similar works. In section IV, we summarize our proposal of modelling the behavior of a player using machine learning. Section V describes the game used as a use case for the validation of the proposal, as well as the modeling used and the game behaviors that are simulated. The results' evaluation is shown in section VI. Finally, some conclusions and the future works is presented in section VII.

II. Theoretical Reference

A. *Finite State Machines applied to gaming*

Digital game developers usually use the same set of techniques in the game's artificial intelligence implementation, which are finite state machines (FSMs) and fuzzy state machines (FuSMs). According to Bourg and Seemann [8], an FSM is an abstraction of a machine that drives several predefined states, defining conditions that determine when a state is present, while the current state determines how the machine behaves. In Fig. 1, it is possible to observe the three behaviors of the ghosts of the original Pac-Man game being represented by states. First, a Scatter behavior, where the ghost moves toward corners and circles, is activated after 20 seconds of hunting. Then, a Hunting behavior, where each ghost has its implementation and depends on how far in the game the player is, is activated after 5 seconds scattering or 10 seconds running away. Finally, a Fleeing behavior, where the ghost moves randomly in a slower way, is triggered when the player captures an energy pellet.

The Pac-man approach uses finite state machine (FSM), which is popular because it requires little power of processing and it is easy and intuitive to define behaviors, mainly with the aid of visual tools. Buckland [9] presents some advantages in the use of finite state machine:

- Ease and speed: There are several ways to program a finite state machine and almost all of them are very simple.
- Ease of Debugging: When an agent's behavior is being interpreted incorrectly, it can be easily debugged by tracing the code between states.
- Flexibility: an agent can be easily adjusted and evolved into a digital game project, being able to be updated with new states and conditions.

However, there are some known limitations, such as combinatorial explosions. As the complexity of the environment increases, so does the number of states and transitions, since the FSM needs to predict all possible cases and situations in the environment. The modeling of a complex behavior may require a large number of states and transitions, resulting in poorly structured, unrealistic, and chaotic diagrams [10]. To deal with this problem, hierarchical finite state-state machines (HFSMs) are used, where each state can be a new FSM. Any hierarchical state machine can be rewritten as a state machine in the presence of the hierarchy [10]. This other approach is quite powerful but does not solve another problem caused by repetitive behaviors, since the FSM has a fixed set of states and transitions and if the same situation occurs more than once, the activated behavior will be the same in both situations.

Fuzzy state machines (FuSMs) were created as an attempt to mitigate the problems of repetitive behavior present in the FSMs. This machine, instead of using traditional Boolean logic, employs fuzzy logic, which can be classified as a super-set of that logic [11]. Over time, it has been observed that fuzzy logic (logic that deals with values ranging from 0 to 1, different from Boolean logic that only supports Boolean values, that is, true or false) presents some advantages, such as synthesis, adaptation, flexibility and versatility that are important in the modeling of artificial intelligence in a game. Despite having some disadvantages, such as the heuristic nature and the potential combinatorial explosion of rules and antecedents that could generate debugging problems and excessive memory and processing consumption, they are quite useful for this purpose. In the Unreal Tournament 2004, an FPS game in which players move within a virtual world, bots are controlled by a fuzzy state machine [12], where elements (weapons and items) appear periodically and the purpose of the game may vary depending on the chosen mode.

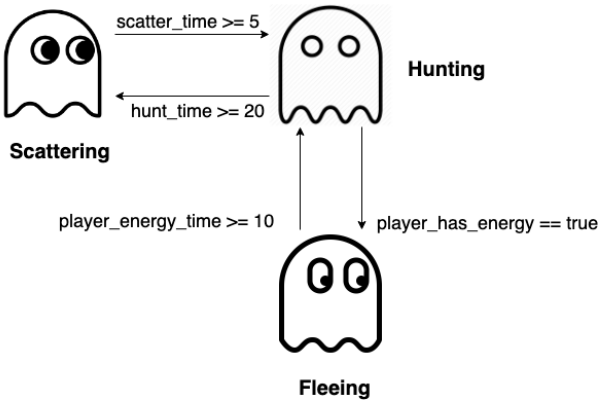

Figure 1. A Possible Pac-Man finite state machine representation.

B. Artificial Intelligence

Artificial intelligence (AI) is the study of "systems that act in a way for any observer to appear intelligent" [13], while including techniques that are used to solve simple problems.

It is divided into three different Artificial Intelligence domains [14]. The first one, the philosophical approach tries to understand the nature of thought and intelligence. The second domain is about the psychology of understanding the human brain and mental processes. The final domain is engineering, where people try to create algorithms to perform human-like tasks.

Russell and Norvig [15] divides Artificial Intelligence into four groups:

- Acting Humanely: Where the system has to interact with people, such as when to explain when you arrived at a certain diagnosis or a system of processing of dialog with user. The reasoning of the system may or may not be based on a human model, different from its behavior, which must be based on a human model.
- Acting Rationally: Where the system acts rationally, trying to achieve the objectives given a certain scenario. With this in mind, it takes the ability to represent knowledge and reason, in order to achieve good decisions in a wide variety of situations. For example, humans need to be able to generate understandable information, in other words, natural language phrases that help humans live in a complex society.
- Thinking Humanly: Where the system tries to think like a human, and for that, it is necessary to know the human mind. Although it is a field widely studied, this function is not yet known, but the field of cognitive science brings together models of artificial intelligence and experimental techniques of psychology to try to build accurate and testable theories of the functioning of the human mind.
- Rationally Thinking: Where the system tries to think rationally, often using logical notation. There are two major obstacles to this approach. First, it is not easy to transform informal knowledge into terms required by logical notation. Second, there is a big difference between solving a problem "in principle" and doing it in practice. Although both obstacles apply to any attempt to build computational reasoning systems, they first appeared in the logical tradition, because the power of systems of representation and reasoning are well defined and well understood.

C. Machine Learning

In machine learning, computers are programmed to learn from past experience [16] and their tasks are typically categorized into three classes:

- Supervised Learning: From a set of labeled data where we already know what the correct output is, the goal is to learn the relationship between input and output data. Supervised learning problems can be classified as regression or classification problems.
- Non-Supervised Learning: From a data set where we have little or no idea of what our results may look like, the goal is to discover new patterns in the data or be a means to an end, such as being applied to problems of clustering.
- Reinforcement Learning: When there is no need for a data set, there can be an agent that must learn how to behave in a dynamic environment through trial and error interactions, and a return with rewards or punishments is provided.

D. Neural Networks

A neural network is a system where modeling is done according to how the human brain performs a task. To achieve good performance, neural networks employ a massive interconnection of simple computational cells, called "neurons" or processing units [17]. According to Nied [18], the model, called the MCP (McCulloch-Pitts) neuron, is described as a set of n inputs to each entry which is multiplied by a certain weight, and then, the results are summed and compared to a threshold. In Fig. 2, it is possible to observe a set of neurons arranged in several layers with weighted interconnections between them.

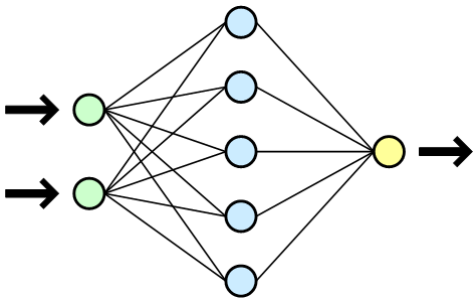

Figure 2. Simplified diagram of a neural network.

According to Braga et al. [19], a basic neural network model has the following components:

- Set of synapses: connections between neurons of the neural network, where each of them has a synaptic weight.
- Integrator: performs the sum of the input signals of the neural network, weighted by the synaptic weights.
- Activation function: restricts the amplitude of the output value of a neuron.
- Bias: value applied externally to each neuron and has the effect of increasing or decreasing the net input of the activation function.

Neural networks are differentiated by their architecture and the way the weights associated with the connections are adjusted during the learning process. The architecture of

a neural network restricts the type of problem in which the network can be used, and is defined by the number of layers (single layer or multiple layers), number of nodes in each layer, type of connection between nodes and its topology [17].

There are several topologies that can be used to build a neural network. Among the best known are Perceptrons and Multi-Layer Perceptrons. Rosenblatt [20] proposed a network topology called MLP (Multiple Layer Perceptrons) where neurons are arranged in the form of a composite network of layers, which enabled an increase of works related to neural networks until 1969.

In that same year, Minsky and Papert [21] showed deficiencies and limitations of the MLP model, causing a lack of interest in studies related to neural networks. It was only after 1982, with the publication of Hopfield's work [22], that the interest in the studies related to neural networks aroused again.

MLP architectures are the most widely used and well-known artificial neural models. An MLP network is subdivided into layers: an input layer, intermediate or hidden layer(s) and an output layer [18]. A multilayer neural network is typically composed of aligned layers of neurons [23]. The input layer distributes the input information to the hidden layer(s) of the network, while in the output layer, the problem solution is obtained. The hidden layers are intermediate layers, whose function is to separate the input and output layers and solve non-linear problems. Usually, the neurons of a layer are connected only to the neurons of the immediately posterior layer, without feedback and connections between neurons of the same layer. In addition, the layers are fully connected.

In further accordance with Santos et al. [23], there are many methods for training a neural network, but backpropagation is the most used one. This algorithm requires the selection of a set of parameters (initial weights, stopping criterion, learning rate, number of iterations of the algorithm), in which its influence can be decisive for the generalization capacity of the network. The learning process consists of two stages: propagation and backpropagation. In the propagation step, an activation pattern is applied to the nodes of the input layer and its effect propagates through the entire network, layer by layer. In the last layer, a set of outputs is produced, configured as the actual network response. In the backpropagation step, all synaptic weights are adjusted according to an error correction rule. The error signal is propagated back through the network against the direction of the synaptic connections and the synaptic weights are adjusted to cause the actual network response to approach the desired response, in a statistical sense [18].

In Fig. 3, it is illustrated the process of backpropagation, where weights are updated accordingly to each node error. *Wi'* and *Wj'* represent the values optimized for the neural network while learning to more efficiently map the inputs to the outputs.

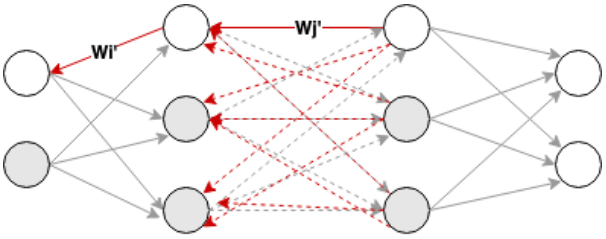

Figure 3. Retropropagation example.

III. Related Works

Tesauro [24] presents a neural network that is trained to be an evaluation function for the game of backgammon, playing against itself and learning from the result, called TD-Gammon. Although TD-Gammon has largely outgrown previous computer programs in its ability to play backgammon, that was not the goal of development. Instead, it aimed to explore some new ideas and interesting approaches to traditional problems in the field of reinforcement learning.

Geisler [25] uses machine learning techniques so that agents learn to play the first person shooter (FPS) style from a data set of a specialist player. The actions of the investigated player were to accelerate and decelerate, direction of movement, direction of the face and moment of leap. Among the techniques investigated, which included Decision Trees, Naive Bayes classifiers and Neural Networks, the models obtained by the Neural Networks were the ones that had more success in the four types of actions investigated, but also those that used more CPU cycles, creating a questioning for the use of this technique in an online game mode.

Muoz et al. [26] presents a controller for a simulated car racing championship. The idea was not to create the fastest driver, but a driver similar to the human. In order to achieve this, a process was first created to build a model of the tracks while the car was running and then several neural networks were used to predict the trajectory that the car should follow and the target speed. Neural networks were trained with data retrieved from a human player and the results showed acceptable performance on unknown tracks of more than 20% slower times than the human on the same tracks.

[27] describes an artificial intelligence system for racing games. Throughout the training, the system observes the positioning of the road, the player, the choice of the race line, the speeds reached along the course and the use of the brake and accelerator. This information is then processed and absorbed into an artificial intelligence model called *Drivatar* that is representative of that player's driving style. Such a model can subsequently be used to dynamically generate a plausible variety of race lines and racing behaviors. It is also important to note that this model is non-deterministic and

therefore will not produce the same output every time, not just a simple recording of the player's direction. In general terms, *Drivatar* learns a model of the player's trajectory based on characteristics of the local topology of the track, such as geometric measurements and a model of the speed of the player depending on the trajectory and some relevant properties of the car being handled. When *Drivatar* is asked to drive in a new lane, the geometry can be inserted into the model as input, and a credible (if not perfect) race line will appear as output. Likewise, the race line generated, along with the relevant car variables, can be provided for the speed function. This will automatically take into account the fact that *Drivatar* may be driving a car much different from the one used in training. Therefore, it does not drive at the same absolute speed (which would seem clearly wrong), but will be proportional to the capacity of the new car, consistent with the training data. Although this is not necessarily realistic in the real world, it makes a lot of sense from a game perspective.

Pereira [28] introduces a new reinforcement learning algorithm, called Pessimistic Q-Learning. Although it does not aim to simulate human behavior, the study tries to solve the problem of generating robots capable of playing turn-based games seeking the best results. The experiments were carried out in different scenarios, being the following traditional games: TicTacToe, Connect-4 and CardPoints. A comparison was also made with the traditional Q-Learning algorithm, where the results illustrate gains in quality in the proposed technique.

Ortega et al. [29] describes methods, based on machine-learning techniques, to generate controllers that mimic the playing style of a given human player, where the player's style is measured through an evaluation structure, which compares the game of a human player with the dotted game path of an AI player. Neural networks with backpropagation and neuroevolution for supervised learning and dynamic scripting were used for reinforcement learning. Having as a validation game a version of the classic platform game "Super Mario Bros", Neural Networks with neuroevolution obtained better performance both in terms of the measure of instrumental similarity and in the phenomenological evaluation by human spectators.

Cardoso [30] uses artificial intelligence, classification and association algorithms along with fuzzy logic, specifically Apriori, FuzzyDT, FCA-BASED, C4.5, PART, and Ripper to extract relevant Blackjack rules. The study attempts to find ways to minimize the casino's edge and turn the odds in favor of the player using gambling strategies.

Mendonça et al. [31] presents two machine learning techniques, a Reinforcement Learning approach and an Artificial Neural Network (ANN), that are used in a fighting game in order to allow the agent/fighter to emulate a human player. The results in the experiments indicate that the proposed methods had better performance against human players than those obtained by AI bots derived from other state-of-the-art methods.

Oshri and Khandwal [32] uses a Convolutional Neural Network (CNN) to make predictions of movement in chess. The task has been defined as a two-part classification problem: A CNN part selector is trained to score which white pieces are to be moved and CNC selectors for each part that produces scores to where it should be moved. The networks were trained using 20,000 games, consisting of 245,000 moves made by well-skilled players. The parts selection network was trained in all these movements, and the motion-selection networks were trained in all movements made by the respective piece. The success of convolutions in the model is reflected in how locally moving parts perform better than those moving globally. The network was played as an artificial intelligence against another artificial intelligence existing for purposes of teaching the game of chess, tying with 26 games in 100 and losing the rest. It was concluded that the convolution layers in the deep learning approaches to chess are useful in recognizing patterns of small local tactics and that this approach should be trained and composed with evaluation functions for a smarter global game.

Table I
COMPARISON BETWEEN RELATED WORK

Reference	Learning Type	Techniques	Use Case	Behavior Simulation
[24]	Reinforcement	Temporal Difference	Gammon	No
[25]	Supervised	Decision Tree Naive Bayes Neural Network	Soldier of Fortune 2	Yes
[26]	Supervised	Neural Network	The Open Racing Car Simulator	Yes
[27]	Not Specified	Not Specified	Forza Motorsport	Yes
[28]	Reinforcement	Q-Learning Q-Learning Pessimist	TicTacToe Connect-4 CardPoints	No
[29]	Supervised Reinforcement	Neural Network Dynamic Scripting	Super Mario Bros	Yes
[30]	Supervised	Decision Tree Fuzzy Logic	Blackjack	Yes
[31]	Supervised Reinforcement	Neural Network Q-Learning	Boxer	Yes
[32]	Supervised	Neural Network	Chess	Yes

IV. MODELING PLAYER BEHAVIOUR WITH MACHINE LEARNING

To reduce the risk of a player losing interest in a game, we propose to use the behavioral simulation of a human player with specific characteristics in order to generate a

better balance between the game-play of the human opponent and the Artificial Intelligence bot simulating their behavior. In order to perform this behavioral simulation, machine learning techniques, such as Neural Networks, may be applied, however, our methodology is not limited to solely these techniques and may be applied with many supervised techniques.

In Fig. 4, it is possible to observe that the the training phase of our proposal uses a dataset built from several games played by a given player, the one that will be simulated. With this dataset, we generate the attributes needed to use in our proposed modeling, which will be used by the machine learning technique as input data. After this training phase, the generated models can be evaluated and analyzed for efficiency and, if necessary, a new attribute selection process.

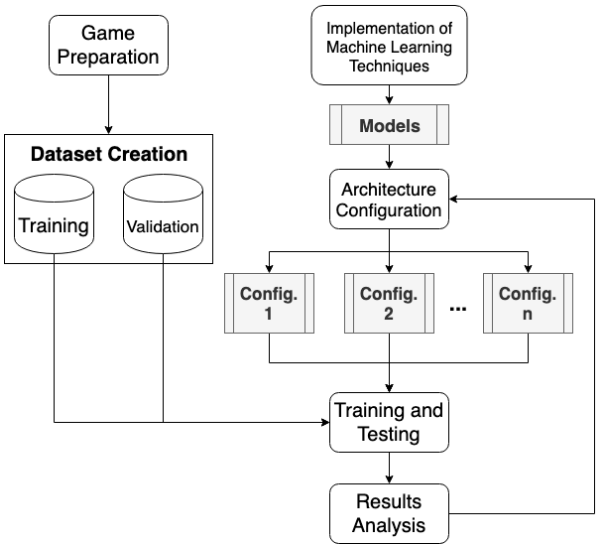

Figure 4. Proposal scheme.

In this context, an investigation will be carried out on the relation between the performance of the classifiers generated from the selection of attributes relevant to the domain. The objective here is to use data from several games of a certain player, in order to generate a model capable of simulating the behavior of this player for a set of inputs never explored before.

V. The Use Case - Hearts Game

In order to evaluate our proposal, we chose Hearts, a card game involving 4 players. From adaptations of an existing game code, it was possible to generate different behaviors for the bots to be simulated and generate a database to be used with the machine learning techniques.

A. Hearts Rules

The game uses a single deck with 52 cards, 13 for each player and ends when a player reaches 50, 100 or 150 points, as configured. The winner is, then, decided based on the player that has the less amount of points. The score is based on the number of cards each player gets at the end of every round. Each card of the Hearts suit is worth a single point, the Queen of Spades is worth thirteen and the Jack of Diamonds is worth ten negative points. If a player collects all points, he is awarded a zero score, while the others receive 26 points each. This is called "Shooting the Moon". To decide the winner of each round, it is necessary to see which is the highest card of the table in relation to the strength and rank of the initial played suit, regardless of whether it is a Hearts or not. When all 13 cards of each player are played and there are no cards left, this hand is finished and the score is evaluated. Before the match begins, each player chooses three cards, and passes them to another player. The main objectives of passing are to try to become "short" (few cards) or "void" (no cards) of a suit, and thus be able to play off-suit when that suit is led, or to get rid of some of the "dangerous" cards that will likely force that player to "win" a round containing penalty points, such as Aces, Kings, or Queens of any suit (especially spades and hearts).

1) Bot Behavior: Five different behaviors were defined for the "pass" movement at the beginning of each match, which changes the order of the player's hand, influencing each subsequent move in the game, as the cards are selected according to this ascending order assigned to the player's hand:

- VOIDER: First, the player's hand is ordered according to the suits that has fewer cards, ignoring their values. Then, it is ordered by the sum of the value of the cards with the weight of the suit, which is the sum of all the cards of a certain suit divided by the number of cards of that suit.
- SHOOTER: First, the player's hand is sorted by the face value of the cards. Then, the hand is sub-sorted by the suit that has fewer cards. Finally, all Hearts cards are put at the end of the sorting.
- SHERIFF: First, the player's hand is sorted by the face value of the cards. Finally, all Hearts cards are put at the end of the sorting.
- BOUNTY HUNTER: All positive point cards are placed at the end of the hand. Finally, the player's hand is sorted by the suits that has fewer cards in hand.
- LOW LAYER: The player's hand is sorted solely by the values of the cards with the weight of the card.

2) Dataset Modeling: The samples obtained from the processing of games played by the bots are saved in a CSV format file. The file fields are exemplified in Table II, where each card in the deck is represented by a column (concatenating the initial letter and the value of the card), and its values are integers that represent the card's location in the current game state, as can be seen in Table III. The

output column contains the value of the card that was chosen by the player given the game state represented by the other columns. These samples are sub-divided in training and validation datasets, that are used to train a machine learning model and to evaluate its performance, respectively.

Table II
REPRESENTATION OF THE GAME STATE USED IN THE TRAINING DATASET

AH	2H	3H	4H	...	TC	JC	QC	KC	Output
0	2	0	0	...	0	0	0	0	2C
0	2	0	0	...	1	0	0	0	AD
0	2	0	0	...	1	0	0	0	4D
1	2	0	0	...	1	0	0	0	7H

3) Neural Network Implementation: The Scikit-Learn library was used for the implementation of the *Multilayer Perceptron* Neural Networks. Several initial experiments were performed with the available parameters in search of the best performance model, such as the number of neurons per layer, number of hidden layers, regularization value, type of optimization and activation function. We highlight the number of iterations and hidden layers, which were the strongest influencing parameters in the final results. In Fig. 5, it is possible to verify that, by increasing the number of hidden layers and the number of iterations, the loss values decreased, although with 7 hidden layers or more, more iterations are necessary to achieve the same results. In Fig. 6, it is possible to observe that by increasing the number of hidden layers, the results improve, both for the training and validation datasets. With 15 hidden layers or more, however, the results start to decrease again. Bear in mind that this is not the final evaluation of the model but an intermediate phase. One bot can play different cards and have the same final result. In the end, the bots are evaluated by full match results and not just by comparing each card chosen in a given scenario.

Table III
REPRESENTATION OF THE CARD STATE IN THE MATCH

Numerical Value	Description
0	Unknown
1	Already used
2	In player's hand
3	In table
4	In opponent's hand, if known

VI. RESULTS EVALUATION

The models generated to perform the result analyzes were trained from a dataset with 9,500 matches versus the same opponents, Sheriff, Shooter and Low Layer. The behaviors for those opponents were randomly chosen but the same ones were used for all 5 bot behaviors. The *gameseed* allows to control the initial state of the game (initial cards in the hands of each player), however from the moment the players make their actual moves, the game can completely change. A *Multilayer Perceptron* Neural Network was used, with 5 hidden layers, each with 52 neurons, alpha value of 1e-02, relu as the activation function and the adam solver.

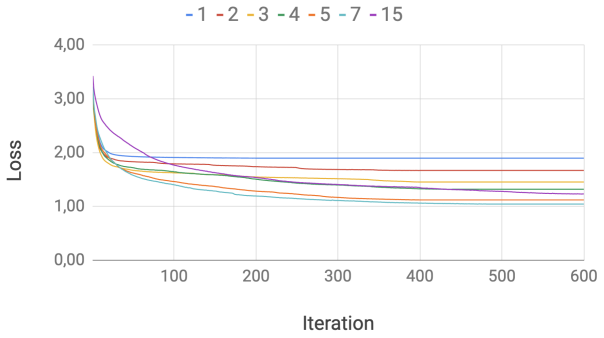

Figure 5. Comparison between loss values by iterations for different amount of hidden layers.

After the training phase, the first validation conducted was the comparison of the responses of each model against the responses of their respective bot for each play in 10,000 matches, all with different controlled *gameseeds* from those used in the 9,500 training matches. As can be seen in the table IV, all models chose the same card as their respective bot in at least 58% of cases. Even so, in up to 79.0% of the cases the card chosen by the respective bot was within the first 3 alternatives of choice of the model, considering that the response of the model consists of a list of cards ordered by probability.

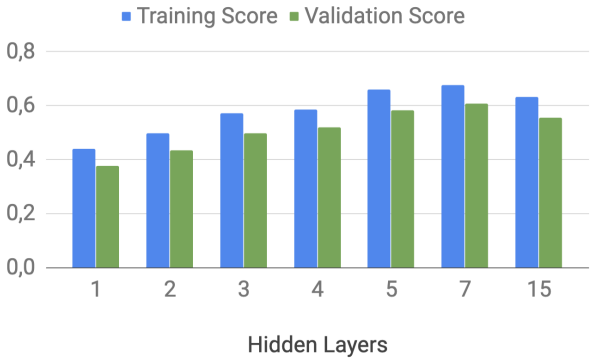

Figure 6. Partial scores of training and validation by amount of hidden layers.

From the data obtained by comparing the response of the model to each play with the bots, the next step was, from the same gameseeds of the 10,000 matches, to let the model play the matches from start to finish and compare their final results. The idea here is to compare if the generated bot

performs the same result, in terms of final score position (victory, 2^{nd}, 3^{rd} or last), as in the original recorded played match. Although the model has not chosen the same cards, all the times, if the training has been successful, the choice must be related to its behavior, ensuring that the final results are likely to be the same.

Table IV
STATISTICS OF THE CHOSEN CARDS BY THE GENERATED MODEL

Model	Plays	Same Card	Inside Top 3	Outside Top 3
Voider	608,738	58.2%	79.0%	21.0%
Shooter	1,315,314	62.3%	81.7%	18.3%
Sheriff	634,829	69.7%	88.0%	12.0%
Bounty Hunter	641,433	59.6%	79.7%	20.3%
Low Layer	1,242,722	66.8%	86.4%	13.6%

As it can be seen in the Fig. 7, all models managed to get close to the amount of victories of their respective bots, keeping also the amount of victorious matches in common (intersection) with a rate higher than matches won only by the model. The stacks represent victories against opponents of the grouped model, the order of behaviors being the same as that of the pooled models (Voider, Shooter, Sheriff, Bounty Hunter and Low Layer). The lighter colors are the representation of the intersection of victories between the bot and its particular model. The intermediate color represents the wins of matches that the model won but the bot lost and the darker color represents the games in which the bot won but the model lost. Therefore, the best results are when the lighter bars are bigger, then the intermediate ones. Finally, the darker bars should be small. For example, the second stack of the "Voider" model shows that on average 86% of the matches against opponents "Shooters" were won, of which 76% on average were the exact same games that the "Voider" bot won. Behavior learning is also clear on the difference of wins depending on opponents, that is, opponents who were difficult to win, in a particular match, remained difficult and easy ones remained easy.

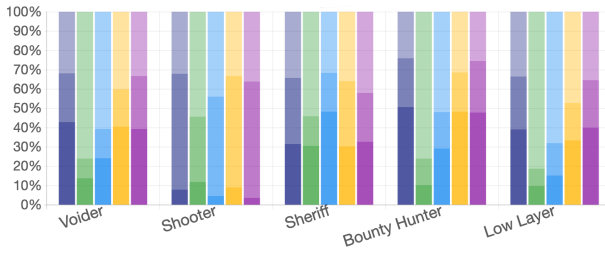

Figure 7. Models result statistics.

Regarding not only victories, the models also showed a similar trajectory to their respective bots with the number of times they were in second, third and fourth place, as it can be compared in the Tables V, VI, VII, VIII and IX.

Table V
VOIDER RESULT STATISTICS

Opponents (1vs3)		Voider	**Shooter**	Sheriff	Bounty Hunter	Low Layer
1^{st}	Bot	2449	**7721**	5826	4043	2730
	Model	1402	**6662**	4430	2404	1657
	Intersection	782	**5884**	35464	1620	909
2^{nd}	Bot	2513	**1843**	2897	2890	2271
	Model	1699	**2397**	3201	2419	1567
	Intersection	676	**877**	1419	963	573
3^{rd}	Bot	2498	**392**	1010	1897	2379
	Model	2446	**810**	1676	2560	2225
	Intersection	825	**153**	422	723	702
4^{th}	Bot	2540	44	267	1170	2602
	Model	4453	**131**	693	2617	4551
	Intersection	1794	**18**	122	728	1830

Table VI
SHOOTER RESULT STATISTICS

Opponents (1vs3)		Voider	Shooter	Sheriff	Bounty Hunter	**Low Layer**
1^{st}	Bot	245	2554	378	240	**404**
	Model	273	2249	361	272	**426**
	Intersection	116	1386	166	120	**230**
2^{nd}	Bot	164	2465	651	157	**255**
	Model	205	2340	614	171	**274**
	Intersection	61	1046	253	54	**102**
3^{rd}	Bot	577	2540	1867	700	**584**
	Model	631	2492	1922	678	**661**
	Intersection	254	1131	952	270	**272**
4^{th}	Bot	9014	2441	7104	8903	**8757**
	Model	8891	2919	7103	8879	**8639**
	Intersection	8485	1615	6141	8360	**8187**

VII. CONCLUSION

The skill level of game participants (simulated people or bots), that we can interact, can be defined by a set of algorithms and heuristics that may be considered efficient for a given public, but inefficient for others. Thus, with the

Table VII
SHERIFF RESULT STATISTICS

Opponents (1vs3)		Voider	Shooter	Sheriff	**Bounty Hunter**	Low Layer
1^{st}	Bot	1657	5196	2582	**1333**	2562
	Model	1137	3615	1338	**931**	1729
	Intersection	567	2808	818	**479**	1077
2^{nd}	Bot	840	2823	2424	**912**	868
	Model	556	2800	1875	**649**	699
	Intersection	137	1181	739	**175**	188
3^{rd}	Bot	1299	1471	2491	**1755**	1258
	Model	1015	2220	2555	**1376**	1028
	Intersection	250	654	915	**499**	316
4^{th}	Bot	6204	510	2503	**6000**	5312
	Model	7292	1365	4232	**7044**	6544
	Intersection	5222	287	1833	**5083**	4408

application of machine learning techniques, it is possible to have behaviors simulated in order to make the game environment more attractive to the player, specially when considering environments where player disconnections are a big concern.

Table VIII
BOUNTY HUNTER RESULT STATISTICS

Opponents (1vs3)		Voider	**Shooter**	Sheriff	Bounty Hunter	Low Layer
1^{st}	Bot	1568	**7288**	5087	2514	1705
	Model	774	**6544**	3608	1306	892
	Intersection	378	**5551**	2650	793	436
2^{nd}	Bot	1911	**2097**	3015	2565	1730
	Model	1139	**2468**	3111	1828	1040
	Intersection	417	**956**	1267	725	347
3^{rd}	Bot	2511	**540**	1436	2501	2387
	Model	2108	**807**	2043	2546	1972
	Intersection	732	**157**	575	901	732
4^{th}	Bot	4010	**75**	462	2420	4178
	Model	5979	**181**	1238	4320	6096
	Intersection	3180	**17**	221	1795	3337

Table IX
LOW LAYER RESULT STATISTICS

Opponents (1vs3)		Voider	**Shooter**	Sheriff	Bounty Hunter	Low Layer
1^{st}	Bot	2227	**7325**	4661	3931	2467
	Model	1361	**6622**	3958	2620	1482
	Intersection	749	**5961**	3179	1861	872
2^{nd}	Bot	2763	**2173**	3545	3106	2465
	Model	2070	**2597**	3697	2849	1908
	Intersection	904	**1336**	2257	1311	763
3^{rd}	Bot	2633	**451**	1430	1938	2538
	Model	2806	**665**	1770	2447	2604
	Intersection	1043	**209**	849	797	954
4^{th}	Bot	2377	**51**	364	1025	2530
	Model	3763	**116**	575	2084	4006
	Intersection	1652	**25**	203	659	1780

In this work, we present a methodology to simulate the behavior of a player using a *Multilayer Perceptron* Neural Network Machine Learning scheme based on gaming history. We evaluate our methodology in a very popular cards game called Hearts. In an effort to evaluate behavioral similarities for each of the presented models, two test metrics are presented:

1) Comparison between the cards chosen by the generated model and its specific bot;
2) Comparison between the final score position (1^{st}, 2^{nd}, 3^{nd} or last) by the generated model and its specific bot.

It is possible to evaluate that neural network hyperparameters and the modeling used on input data influence the quality of the obtained results. All models were able to get closer to the results of their respective trained bots. Through the evaluation of the results of 10,000 matches, with different *gameseeds* of the 9,500 matches used for training, the best results are for the model Shooter versus 3 Sheriffs getting 95.5% in the amount wins compared to their particular bot and the worst result was for the Bounty Hunter model versus 3 Voiders getting 49.36%. Besides that, another contribution is the generated gaming history dataset, both for testing and for validation. We also adapted an existing Hearts game code to allow the simulation of such behaviors addressed in this work.

By analyzing the experimental results, we showed very good results for the proposed technique that is able to provide a simulation of player behaviour in games. Nevertheless, improvements always can be incorporated, and tests should be done with real players, not bots with certain behaviors. Real players usually do not have a single behavior and can change their strategy as the game progresses, so it is also a good use case to validate the learning of different behaviors for specific players. In addition, the proposal may be validated using other machine learning techniques, specially deep learning techniques where we expect even better results. We also plan to test our methodology in other more complex games like Chess and Canasta[1].

ACKNOWLEDGEMENT

This material is based upon work supported by Air Force Office Scientific Research under award number FA9550-19-1-0020.

REFERENCES

[1] R. Penaforte, "Mercado de games cresce 15% no brasil," 2019 (accessed February 16, 2019). [Online]. Available: https://www.otempo.com.br/interessa/tecnologia-e-games/mercado-de-games-cresce-15-no-brasil-1.2137277

[2] J. Batchelor, "Global games market value rising to $ 134.9bn in 2018," 2018 (accessed February 16, 2019). [Online]. Available: https://www.gamesindustry.biz/articles/2018-12-18-global-games-market-value-rose-to-usd134-9bn-in-2018

[3] W. Barendregt, "Evaluating fun and usability in computer games with children," Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, 2006.

[4] D. Takahashi, "How microsofts turn 10 fashioned the a.i. for cars in forza motorsport 5 (interview)," 2013 (accessed September 27, 2018). [Online]. Available: https://venturebeat.com/2013/11/06/how-microsofts-turn-10-fashioned-the-ai-for-cars-in-forza-motorsport-5-interview/

[5] M. McFarland, "Google's ai just beat the world's best go player," 2017 (accessed September 28, 2018). [Online]. Available: https://money.cnn.com/2017/05/25/technology/alphago-china-ai/index.html

[6] S. Gibbs, "Alphazero ai beats champion chess program after teaching itself in four hours," 2017 (accessed September 28, 2018). [Online]. Available: https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours

[1]Canasta is a popular card game in Brazil and Uruguay where the main goal is to make sets of seven sequential cards of the same suit. Some variations of Canasta in Brazil are *Buraco* and *Biriba*

[7] P. Guilherme, "69% dos internautas brasileiros jogam games eletrônicos, segundo pesquisa," 2016 (accessed October 01, 2018). [Online]. Available: https://www.tecmundo.com.br/video-game-e-jogos/104355-69-internautas-brasileiros-jogam-games-eletronicos-segundo-pesquisa.htm

[8] D. Bourg and G. Seemann, *AI for Game Developers: Creating Intelligent Behavior in Games*, 1st ed. O'Reilly Media, 2004.

[9] M. Buckland, *Programming Game AI by Example*, 1st ed. Jones and Bartlett Publishers, 2004.

[10] D. Harel, "A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[11] L. G. M. Alvim and A. J. de Oliveira Cruz, "A fuzzy state machine applied to an emotion model for electronic game characters," *IEEE International Conference on Fuzzy Systems*, pp. 1956–1963, 2008 (accessed June 19, 2019). [Online]. Available: https://ieeexplore.ieee.org/document/4630637

[12] A. Esparcia, A. Mora, A. Martnez, and P. Garca, "Genetic evolution of fuzzy finite state machines tocontrol bots in a first-person shooter game," *Genetic and Evolutionary Computation Conference*, vol. 1, pp. 829–830, 2010 (accessed October 13, 2018).

[13] B. Coppin, *Artificial Intelligence Illuminated*, 1st ed. Mississauga: Jones and Bartlett Learning, 2004.

[14] I. Millington and J. Funge, *Artificial Intelligence for Games*, 2nd ed. San Francisco: Morgan Kaufmann Publishers, 2009.

[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2009.

[16] K. Faceli, A. Carolina, J. Gama, and A. de Carvalho, *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*, 1st ed. LTC Editora, 2011.

[17] S. Haykin, *Redes Neurais - Princípios e Práticas*, 2nd ed. Bookman, 2001.

[18] A. Nied, "Treinamento de redes neurais artificiais baseado em sistemas de estrutura variável com taxa de aprendizado adaptativa," Ph.D. dissertation, Universidade Federal de Minas Gerais, Minas Gerais, 2007 (accessed October 15, 2018). [Online]. Available: https://www.ppgee.ufmg.br/documentos/Defesas/669/Tese_Ademir_Nied.pdf

[19] A. de Pádua Braga, A. P. de Leon F. de Carvalho, and T. B. Ludermir, *Redes Neurais Artificiais: Teoria e Aplicações*, 2nd ed. LTC Editora, 2007.

[20] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[21] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[22] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science of the USA*, vol. 79, 1982 (accessed October 15, 2018). [Online]. Available: https://bi.snu.ac.kr/Courses/g-ai09-2/hopfield82.pdf

[23] A. M. dos Santos, J. M. de Seixas, B. de Bragana Pereira, and R. de Andrade Medronho, "Usando redes neurais artificiais e regressão logística na predição da hepatite a," *Revista Brasileira de Epidemiologia*, vol. 8, no. 2, 2005.

[24] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, 1995 (accessed October 23, 2018). [Online]. Available: https://cling.csd.uwo.ca/cs346a/extra/tdgammon.pdf

[25] B. Geisler, "An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game," Master's thesis, University of Wisconsin, Madison, 2002 (accessed November 15, 2018).

[26] J. Muoz, G. Gutierrez, and A. Sanchis, "A human-like torcs controller for the simulated car racing championship," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Aug 2010, pp. 473–480.

[27] Vagamelabs, "Drivatars and forza motorsport," 2010 (accessed November 23, 2018). [Online]. Available: https://web.archive.org/web/20100510055332/http://www.vagamelabs.com/drivatars-trade-and-forza-motorsport.htm

[28] A. B. Pereira, "Q-learning pessimista: Um algoritmo para geração de bots de jogos em turnos," Master's thesis, Pontifícia Universidade Católica, Rio de Janeiro, 2012 (accessed October 23, 2018). [Online]. Available: http://www.dbd.puc-rio.br/pergamum/tesesabertas/1021752_2012_completo.pdf

[29] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.

[30] K. R. Cardoso, "Um estudo sobre a definição de estratégias para o jogo de blackjack usando técnicas de aprendizagem de máquina e sistemas fuzzy," Master's thesis, Associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, Mossoró, 2015 (accessed October 23, 2018), 23 out. de 2018.

[31] M. Mendonça, H. Bernardino, and R. Neto, "Simulating human behavior in fighting games using reinforcement learning and artificial neural networks," 2015 (accessed June 23, 2019). [Online]. Available: http://www.sbgames.org/sbgames2015/anaispdf/computacao-full/146986.pdf

[32] B. Oshri and N. Khandwal, "Predicting moves in chess using convolutional neural networks," 2015 (accessed November 23, 2018). [Online]. Available: http://cs231n.stanford.edu/reports/2015/pdfs/ConvChess.pdf