Mimic Players Behavior Using Q-Learning

Luiz Antonio Lima Rodrigues* Paulo Ricardo Bueno † Ricardo Inácio Álvares e Silva‡
Mario Henrique Akihiko da Costa Adaniya§

Centro Universitário Filadélfia, Computer Department, Brazil

ABSTRACT

The study and development of intelligent agents is important to industry due to the fact that they can be used in a range of applications, such as: simulating the human behavior in games of various genres; testing tools and; generating content, e.g maps or cities. This paper presents the generation of artificial intelligence models which mimic the human behavior. Our approach for this generation is to use an approximate agent with Q-Learning, a technique of reinforcement learning. A 2D rogue-like game was used as testbed, which the user goal is to survive the maximum number of days, and it needs to collect food to not starve, and avoid enemies to not get killed. Our aim is to create a model online collecting data during the gameplay. As it do not require the user to answer a questionary or some kind of inconvenient input, or even pause the game this approach is transparent for the player. Furthermore, we present a discussion of the gameplay's performance of the generated model, demonstrating its mimic's hability and generalization. The results demonstrates that the accuracy was 37.04% for the testing sample and 43.5% for the training sample. However our agent reaches its goal, acting with a similar human behavior; having score similar to the human players and; completing the levels with a smaller time on average.

Keywords: procedural content generation, reinforcement-learning, player modeling.

1 INTRODUCTION

The development of a game is a complex task that requires extensive knowledge in order to produce a good work. With the evolution of graphics and hardware, one of the main goals is to attract the players using computational resources that helps in the generation of content [5]. Procedural Content Generation (PCG) is a confident tool used both online and offline [8] to create content like maps [3] and characters [6], decreasing the cost and time of a development team that is creating a game [1].

One of the issues where the artificial intelligence is applied satisfactorily is to hold the player's game interest. In practice there are players who learn faster than others, making the experience relative, causing an intervention by the level's algorithm generator to all kinds of people [10]. Without this, the engagement in the game may decrease, making the player feel bored or frustrated.

In Yannakakis *et al.* [4], Player Modeling consists in the study of Artificial Intelligence (AI) techniques to build player models. This can be done through detecting behavioral, affective and cognitive patterns. Some of the studies presented ways to simulate this behavior using techniques such as reinforcement learning and neural network. Also, comparisons were made from the utility and performance of each technique.

When working with human behavior, it is important to check the level design balance to keep the player motivated to progress through the game and face the challenges. The work presented in Silva *et al.* [12] was introduced as an alternative to balance the difficulty of the game as the player advance, called Dynamic Difficulty Adjustment (DDA). According to the player's skills, the agent was adapted in order to adapt their level of gameplay in levels similar to the opponent. As our work, actions taken by the player will directly influence the Intelligent Agent (IA), which whether he reach an objective, this action will be considered good. Similarly, if your hero is defeated, he will suffer losses and know that this is a negative action to take.

Another way to adjust the game difficulty is presented in Prez *et al.* [10], where the technique called Evolutionary Fuzzy Cognitive Maps (E-FCM) is apllied, a modeling tool based on Fuzzy Cognitive Maps. The difference here is that each state is evolved based on external causes.

There are several techniques to create an IA, where some of them are more effective than others according to the game style. A fighting game called "Boxer" was presented in Mendona *et al.* [9], which demonstrated the use of intelligent agents created from two approaches, the first was based on reinforcement learning using Q-Learning algorithm, able to make an agent simulates human behavior through a set of conditions and actions. The second used an Artificial Neural Network (ANN), which was trained through the history of several fights made by human players. In the training period, the agent using Q-Learning had a better performance, while in the testing period the agent ANN got a better response of most players, due to be more challenging and fun.

Another approach using player modeling is presented in Carvalho *et al.* [2], where the goal was to create an endless game with a map that is considered playable while maintaining an appropriate level balance for the player. It use chunks as obstacles in the map, and to control the production of that, two neural networks was used. The first receives controllable features as input, and the other receives both non-controllable and controllable features as input. The chunk difficulty is the output of the networks. After available, the second network it is used to make adjustments to the game in order to reclassify the chunks. Thus, as more data available, more chunks with real difficulty will be perceived by the players.

In Holmgard *et al.* [6] the Q-Learning was used to produce the agent of the game sample. This technique was selected because any reinforcement learning satisfies the game requirements. Besides, the use of a lookup table, which contains all the Q-Values, was a good idea, due to the small game world and the limited player actions. Each state of the table contains all the game world and the player's hitpoints.

This work aims to present a way to mimic the human behavior through the use of reiforcement learnign technique. We use *Q-Learning* with an approximate agent approach using features. We collect data through 84 days of the game, where volunteers played to improve the diversity that each of them had some level of difficulty and take some time to think and act in the levels. Finally, we

*e-mail: luiz_rodrigues17@hotmail.com
†e-mail: paulorb@edu.unifil.br
‡e-mail: ricardo.silva@unifil.br
§e-mail: mario.adaniya@unifil.br

discuss the results obtained from the agent compared to players and presents the conclusion.

2 METHODOLOGY

This section describes the conducted steps in this research. We present the simulation environment; data collection process; Artificial Model (AM) building and evaluation tests.

2.1 Data Collection

The data gathering was made during a gameplay session, where an individual was asked to play until it death or survive all days. This process was repeated for only 6 students who were willing to collaborate, generating a sample of 84 days. We split it in two samples, one with 70%, the training sample (In_s), and the testing sample (Out_s) with the later.

For each day played during all sessions we stored: the player's score, with respect to Table 1; which day was; the time to complete the level; if the player survived or not and; the set of actions taken. This process was also applied collecting data from game sessions where our generated model played, in order to compare it to the human player's game sessions.

2.2 Testbed Game

A 2D rogue-like game was adopted as testbed to evaluate our approach. This game is based on an Unity's tutorial and uses its graphical resources, which are available for free download[1]. The goal is to survive the maximum number of days as possible, in a zombie apocalypse.

A day is represented by a single level, in which the player must reach the exit to complete it. The game is turn based, where all enemies move once after the player moves twice. Each step decrements the player food and being attacked by an enemy decreases it by 10 or 20, while collecting a fruit or soda increase by 10 or 20. Other hazards, such as sticks (breakable) and rocks (non-breakable) might be found in a level.

A range of 15 days were stipulated. At the end of each day, the player score is calculated by the sum of rewards from each step taken. Table 1 demonstrates each situation's score. These levels were created with the aim to increment the challenge provided, increment gradually the difficulty by populating the map with elements that induce the action of the players.

Table 1: This table demonstrates the rewards received at each game's situation.

Situation	Step	Z1	Z2	Apple	Soda	Die	Win
Reward	-1	-10	-20	10	20	-100	300

2.3 Artificial Model Generation

This section describes the process used to generate the artificial model. First, we briefly describes the learning algorithms and the agent's type we used. Finally, the features selected and model generation applied are presented.

2.3.1 Q-Learning

The *Q-Learning* algorithm is a reinforcement learning technique where the agent learns through its past actions, trying to maximize the sum of rewards it receives [15].The algorithm build up the Q-Table through the environment exploration using an adapted Bellman's equation which is mathematically represented as:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max(Q(s',a'))) \quad (1)$$

[1] unity3d.com/pt/learn/tutorials/projects/2d-roguelike-tutorial

The particularity of this algorithm is that its table stores the Q-Value for each action A which is allowed in a state S. Furthermore, it is argued that if both learning exploration rate decreases as required from 1 to 0, it will converge to the optimal policy.

After try an action in a state, the agent evaluate the state that it has lead to. If it has lead to an undesirable outcome the Q value of that action from that state is reduced, so that other actions will have a greater value and will be chosen, for the next time it is in that state. Similarly, if the agent are rewarded for taking a particular action, the Q-value of that action for that state is increased, so it is more likely to choose it again the next time if it is in that state. Thus, the goal of an agent is to maximize the total reward.

It is important to notice that when updating Q-value, the algorithm is updating it for the previous state-action combination. After the agent see the results, the Q-value can be updated.

2.3.2 Approximate Agent

The artificial model that we generated to mimic the players uses an approximated agent [11], which is an agent that abstract the main features of the environment. This approach improves the Q-Learning because it allows to represent the pertinent aspects from a state through the weight attributed to each feature, which represents the learning value of that state. Thereby, there is no need to store the Q-Table.

While the approximate agent assists in the state of abstraction and thus the Q-Learning, it requires a high computational complexity due to the fact that it is necessary to define the vector containing all the features values. The biggest challenge in this case is to define these values in order to find acceptable results in accordance with the proposal of the game.

Given a state s, the action is chosen by the following equation

$$A(s) = \arg\max_a F[s,a] * W[s,a] \quad (2)$$

where a is the set of possible actions, *F[s]* is the vector containing the sign of each feature and *W[s]* represents the weights vector, both which respect to state s.

2.3.3 Features

All features calculates the cost from a position X to a position Y using an A* algorithm, with Manhattan Distance as heuristic. The sign of each feature is calculated using the equation

$$s(x) = \frac{c}{cost[x]+1} \quad (3)$$

where $cost[x]$ returns the cost from the player position to element's position represented by x, e.g. the exit tile. The sign function aim is to maximize the sign as the player approximates to its goal.

The feature E captures the main goal of a day, where E represents the exit tile. Min_Z and $Mean_Z$ covers the cost to the enemy which is closest and the mean cost from all enemies. Similar, Min_F and $Mean_F$ have the same aim but with respect to both apples and sodas. E is missing on the extracted features only if the player is corned from enemies. The zombie and food features are generated always that at least one of them exist in the level, at the moment of extraction, e.g. chosing an action. However, when a state leads to the exit position the zombie features will not be generate, once that they do not have relation with the winning state.

E uses $c = 0.05$ and the others $c = 0.1$. This avoids the feature to diverge too much from the others, once that it is the single feature extracted; surviving a day invoke its highest sign and; gives the largest reward.

The cost function ignores the fact that a zombie or soda on the path might influence in its cost. It only considers enemies non traversals position, such as rocks; both food as regular traversable positions and; the cost per action is 3 when it moves over a stick

and 1 otherwise. Note that enemies and foods are captured in its own features, distinguishing when a state is closest to an element and when it is more surrounded, having a high mean.

2.3.4 Model Generation

The model was generated through a training process using a learning rate equals to 0.02 and discount of 0.9; the number of training episodes was 55.

This approach differs from the general Q-Learning because it does not follow the exploration process. It uses data gathered from the players, allowing our model to learn how the players acts, using online learning. This enables the model to absorb knowledgement from all players and abstract their behaviour in the environment, enabling it to act in a similar way to them. Table 2 display the learned weights.

Table 2: This table demonstrates the weights from each feature learned during the training process.

Feature	Weight
Bias	31.2758617
E	1.6145191
Min_Z	-0.07409985
$Mean_Z$	-0.0466128
Min_F	0.2423068
$Mean_F$	0.0238249637

2.4 Evaluation

In this section we describe the test we used to evaluate our generated model and the motivation. All tests were made for both In_s and Out_s, measuring the agent's generalization, besides its capacity to mimic players.

Prediction was evaluated replicating the players' actions and comparing with the action our model chosen in the same state.

Level Evaluation might be used as fitness function, on level generation, applying artificial agents to simulate a real player gameplay, assessing the challenge it should provide to a regular player. We compare the similarity of score our model received to the human players' mean score.

Efficacy has the aim to evaluate the time our model took to complete a level compared to real players. The agent time was compared to the mean time humans took, for each day.

Efficiency was used to check the hability to win each day for our agent compared to the other players.

3 Results and Discussion

In this section we present and discuss the results about our generated model reached after the tests we described in 2.4.

In this test we evaluated the similarity of the rewards our model received, after completing a level. Table 3 demonstrates detailed data about the performance from both In_S and Out_S, beyond our model. All values were normalized using min max normalization, between the values from all samples. The minimum (-388) value came from In_S, when a player get corned by 2 enemies and; maximum (359) from Out_S.

Table 3 demonstrates that our model was able to reach a slight better performance from both samples on average. Also, it demonstrates that the model's standard deviation was smaller. However, this is due to our agent won each testing episode, despite that the largest sum of rewards was received from the other samples. Otherwise, in both samples some individuals died without finishing the last day.

Table 3: This table demonstrates statistic's data from the rewards received during the game play from our model, In_S and Out_S .

–	*AI*	In_S	Out_S
Mean	0.9105849	0.9066569	0.8991368
Std	0.05751841	0.1325192	0.1759487
Min	0.7416332	0.0000000	0.008032129
Q1	0.8982597	0.8995984	0.899598394
Median	0.9196787	0.9330656	0.925033467
Q3	0.9357430	0.9504685	0.961178046
Max	0.9973226	0.9866131	1.000000000

Using an intelligent agent to evaluate a level is a well proposed approach [7, 14, 13]. We argue that the agent generated using the methodology we proposed would be feasible for this purpose. On average, its score was similar to both samples and it was able to finish all levels, despite the use of a small data set.

This test, similar to the Score, investigates the artificial model's performance against the other samples. The values of Table 4 were normalized using the minimum of 12 seconds, reached by our model and; maximum of 67 seconds, where player in both In_S and Out_S samples took.

Table 4: This table demonstrates statistic's data from our model, In_S and Out_S collected during their gameplay with respect to the amount of time they took to complete the levels.

–	*AI*	In_S	Out_S
Mean	0.2250784	0.221157	0.4432602
Std	0.1591309	0.2067062	0.27857
Min	0.0000000	0.01818182	0.1090909
Q1	0.1090909	0.10909091	0.2181818
Median	0.1636364	0.14545455	0.3636364
Q3	0.3454545	0.21818182	0.6000000
Max	0.5636364	1.00000000	1.000000000

On average, our agent and the In_S players almost took the same time, with a slight advantage to our model, while the Out_S player took about twice the time. Despite the several A* searches our model realizes, extracting the state features, the time it took to complete the levels. We argue that the human player took more time due to the game being turn based. Most players took time thinking in which action they should take, while the agent always follows the same algorithm. This is demonstrated in Table 4 through the standard deviation.

The Q-Learning algorithms uses the discount factor to control how much a successor of the state the agent intends, in order to go influence on its utility. However, it is common the human players plan more steps ahead than the agent is able. This is a limitation on our model, which might increase the performance, as a similarity to the players that it is modeled. Furthermore, taking less time to decide the action it will take gives an interesting insight on using this model as utility function together with a Min-Max or Expect-Max, for example. This algorithm would increase the time to choose an

action, in exchange of being able to plan its actions looking more than 2 steps ahead.

To evaluate how the action's selection from our agent mimics the players, we replicated each action the player took, in the same level, and also choose an action with our model. In comparison to the Out_S, it had an accuracy of 37.04% and 0.1108783 as standard deviation. With respect to In_S, our agent had 43.5% of accuracy and 0.1567165 as standard deviation. Table 5

This demonstrates that our approach was not able to mimic the players' action by action. However, this does not confirm that our approach is irrelevant to the purpose of player's mimicking, once that our aim is to abstract the player's behavior, and not to repeat their actions.

The agent settings can be configured with simple but also complex features, using factors that can lead to greater learning. Examples like [12, 9] demonstrates that this kind of technique can generate reliable results, creating the agents that will be used as opponents in games of different genres, like fight and strategy. Our technique might be used in games of other genres, 2D or 3D .However, its perfomance relies on the selected features and their extraction from the environment.

We argue that this high error index is due to our limited data base, reducing the number of episodes available to train. That fact is given to all features used are numerical and we stipulate them based on the testbed game, however, using another set of features might improve its performance. Furthermore, the fact that human player's mostly takes an action based on future steps also contributes to the low accuracy.

Table 5: This table demonstrates statistic's information with respect to the accuracy our agent had, when the action it selected was compared to the actions taken from both In_S and Out_S, at the same state.

–	In_S	Out_S
Mean	0.4350518	0.3704231
Std	0.1567165	0.1108783
Min	0.1578947	0.2162162
Q1	0.3288288	0.3636364
Median	0.4117647	0.3636364
Q3	0.5000000	0.4400000
Max	0.9411765	0.6071429

4 Conclusion

This paper presented a methodology for procedural generation of Artificial Agent models, using reinforcement learning. As proof of concept, we used a rogue-like game, where we collect data from the gameplay of 6 players, generating a data set of 84 days, which we splited 70%-30% for training and testing.

We stipulate a set of features based on a rogue game main goals, e.g. avoid enemies and reach an exit tile. The training process was guided by the players actions, using online learn. This approach avoids the necessity of stopping the player during the gameplay. Also, it does not require to capture any player information or image.

The results demonstrate that our generate agent was able to complete all 15 levels, while most players also did, but sometimes they got stuck by enemies or died with no food. Thus, it was able to reach a mean score which differentiates less than 0.02 for both In_S and Out_S. Also, on average, our model finished the level's in a similar time to the In_S and faster than Out_S. However, when the agent was exposed to the same state that a human, it selected the same action 37.04% and 43.5% for Out_S and In_S respectively.

Our generation process reach our goal, being able to receive scores approximated from the one's the human players had. We argue that its low accuracy is due to the small data set and features selection. Also, the fact that the agent cannot choose its actions based on several steps ahead, such as was noted during the human players' gameplay, and reinforced by the time they took to finish the levels, has influence on the high error index.

We believe that this paper gave interesting insights from how to continue this research and pretend to expand this work, investigating which features reaches the best performances. Also, improving the action's selection algorithm, using an adversarial search, such as Min-Max or Expect-Max, to allow our agent to look more steps ahead. Furthermore, testing it with a learning rate different of 0 might increase the similarity to a real player, causing it to constantly adapts itself to the environment.

Acknowledgements

We would like to thank Fundação Araucária for supporting this project.

References

[1] D. Carli, F. Bevilacqua, C. Pozzer, and M. d'Ornellas. A survey of procedural content generation techniques suitable to game development. In *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*, pages 26–35, Nov 2011.

[2] L. V. Carvalho, Á. V. Moreira, V. Vicente Filho, M. T. C. Albuquerque, and G. L. Ramalho. A generic framework for procedural generation of gameplay sessions. In *SBGames 2013 - Computing Track ()*, 2013.

[3] G. de Oliveira Barbosa Leite and E. S. de Lima. Gerao procedural de mapas para jogos 2d. In *SBGames 2013 - Computing Track ()*, 2013.

[4] D. L. G. N. Yannakakis, P. Spronck and E. Andre. Player modeling, 2013.

[5] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1):1:1–1:22, feb 2013.

[6] C. Holmgaard, A. Liapis, J. Togelius, and G. N. Yannakakis. Generative agents for player decision modeling in games. In *Poster Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.

[7] A. Liapis, G. N. Yannakakis, and J. Togelius. Towards a generic method of evaluating game levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*, 2013.

[8] A. Machado, E. Clua, and B. Zadrozny. A method for generating emergent behaviors using machine learning to strategy games. In *Games and Digital Entertainment (SBGAMES), 2010 Brazilian Symposium on*, pages 12–18, Nov 2010.

[9] M. R. F. Mendona, H. S. Bernardino, and R. F. Neto. Simulating human behavior in fighting games using reinforcement learning and artificial neural networks. In *SBGames 2015 - Computing Track ()*, nov 2015.

[10] L. J. F. Prez*, L. A. R. Calla*, L. Valente*, A. A. Montenegro*, and E. W. G. Clua*. Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps. In *SBGames 2015 - Computing Track ()*, nov 2015.

[11] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[12] M. P. Silva, V. do Nascimento Silva, and L. Chaimowicz. Dynamic difficulty adjustment through an adaptive ai.

[13] G. Smith and J. Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 4:1–4:7, New York, NY, USA, 2010. ACM.

[14] J. Togelius, T. Justinussen, and A. Hartzen. Compositional procedural content generation. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, PCG'12, pages 16:1–16:4, New York, NY, USA, 2012. ACM.

[15] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.