Application of Q-Learning in a Digital Game

Guilherme de Quevedo Brendel and João Ricardo Bittencourt
Escola Politécnica
Universidade do Vale do Rios dos Sinos (UNISINOS)
Porto Alegre, Brazil
guilhermebrendel@gmail.com, joaorb@unisinos.br

***Abstract* — Understanding the use of Machine Learning during game development is essential for the implementation of intelligent agents. In this work the use of reinforcement learning is evaluated, going through details of the implementation of the Q-Learning technique, to the evaluation of the use of it in a digital game.**

Keywords: AI; Reinforcement Learning; Q-Learning; Games;

I. Introduction

As the game industry evolves, it provides us with ever more complex games, with increased production and gameplay value, the application of artificial intelligence algorithms needs to keep up with that, if in the past the agents that were interacting with the players only needed to reproduce especifics pre programmed behaviours, based on rigids state machines, today it's expected that those entities can make decisions and be able to present themselves to the player as real and intelligent beings, this perception from the player is very important to keep the immersion in the game world. Machine Learning Techniques allied with classic algorithms, have proven to be a powerful solution in the field of game development, given its adaptive capacity.

This paper was written in the academic activity of Artificial Intelligence for Games in the course of Digital Games of UNISINOS. In it, it's proposed the application of a Reinforcement Learning Algorithm, Q-Learning, in a context of a digital game.

II. What is Artificial Intelligence?

There isn't a totally accepted definition about what artificial intelligence is, but in a simple way, we can say that it is algorithms of generalization and learning that tries to simulate the human ability to think. Millington (2006) defines artificial intelligence as a technique to make computers capables of thik tasks as humans and animals do. So, to make those computers able of generalization, we have to put then under a session training. Those training occurs to try and error, the role of the algorithms is to provide a way to the computer internalize everything that it learned on a training session.

(Quality Value), in every interaction of the agent with the environment.

III. Reinforcement Learning

According to Sutton and Barto(1998), the problems of reinforcement learning involves learning what to do (through a map of actions to situations), and maximize a numerical reward. Reinforcement Learning is the name that are given to various techniques that generates learning through reward and punishment, an intelligent agent learns to interact with an environment, so he considers his actions and opts for one, those actions alter the environment . Every action is performed with a reward in mind, the agent needs to follow a politic, a series of rules that tries to maximize his score. A model of reinforcement learning algorithm receives a state a an action as input, the output that it generates is a score value, this step will only take the agent to the next state, and, in this way, a new score will be provided for each action. There are many ways to implement a artificial intelligence algorithm, each of them has its peculiarities, and you need to choose the one that best fits the problem you're trying to solve.

IV. The Markovian Decision Process

Andrei Markov was a Russian mathematician, responsible by several studies in the area and specially famous for develop Markovian Decision Process, also known as Controlled Markov Chain, as described by Mitchell (1997). In a markovian decision process the agent must be aware of his current state (S), and from it he has access to a series of possible actions (A), each discrete time stamp (t), the agent executes one of those actions, which brings him to the next state of time (t+1). The actions performed by the agent change the environment in which it is inserted in some way, this change is judged and then a reward(R) is attributed to the agent, $Rt = R(St, At)$.

V. Q-Learning

There are many ways in which you can implement reinforcement learning algorithms, but the most used in games and applications it's the Q-Learning, due to its simplicity of implementation and understanding (Millington, 2006). The Q-Learning algorithm consists in a function of gain that updates the value of Q

As said before, there's a map of the possible actions of the intelligent agent, given a certain state, in which he can choose the action with the greatest associated

value, or a random action (in the implementation you will define how random you want the algorithm to be), it's important to leave a small randomness in the decisions so the agent will always experiment other options and will not fall in a convergence addiction before he found the optimized state. After he has made a decision, the value of Q to that action, in that particularly state will be updated using a gain function, defined by:

$Q = Q + \alpha * (R + \gamma * (maxQ - Q))$.

Q - Represents the current quality state.

α = A value between 0 and 1 that will be multiplied by the reward, it measures how much the Q value will be affected by the action, a 0 value represents no gain, while a 1 value represents maximum gain.

R = The value of the reward for the taken action, this value will vary according to the implementation, it will be a positive value for a considerer good action, and a negative value for bad actions.

γ = The gamma value, is almost like alpha, when it comes to define a multiplication factor, but while the alpha measures the current value, gama will measure the future value if that action is chosen, it's also represented by a value between 0 and 1.

maxQ = The Q value of the future state.

VI. The Training Environment

To apply the Q-Learning the game, Table Heroes, was chosen as training environment, this game is under development using Unity 2017.2, originally a turn-based tactical combat game, where two teams fight each other taking actions on their turns, every unit in the game have a move and an attack action for turn. It's possible to move and to attack, the attacks causes damage to the enemy, the battle ends when all the members of a team are defeated.

Some of the characteristics of the game where changed in order to to get more out of the technique, the test version, then, are composed of:

- A n x n Tile Matrix (The values of "n" can be setted on the implementation).
- An Actor to be trained, capable of execute the following actions:
  - Walk (4 directions).
  - Attack (4 directions).
- 3 enemies.
- A Destination Tile that serves as a objective, the agent reaches the victory state when he enters that tile.

It's possible to manually set the values of the matrix. The Actions of the Actor can occur in any of the 04 adjacents directions, it's not possible to move diagonally, remaining a search space of 8 possible actions. The current state of the actor is defined by a set that takes in account the position [x, y] of the actor (the current tile), and if there's or not a adjacent enemy, so the total number of states is equal to twice the number of tiles. The enemies occupies fixed positions and automatically defeat the actor if he tries to enter their tile. The objective is to reach the green tile.

VII. Agent Architecture

Although it's possible to pass complete info about the world state to the agent, their sensors were propositaly limiteds, so the agent can learn with his experiences. When a new match is started the agent knows his position in the matrix, a two positions vector, he also knows the answer to question "There's a enemy adjacent to me?", if this statement is true, it means hac attack or be attacked, note that, while he knows about the existence of a adjacent enemy, he doesn't know the exactly position. The agent knows his eight possible actions, but the attack actions will only be considered when near to an enemy.

A. The training cycle

A intelligent agent, the target tile (the agent's goal), and three enemies are instantiated in different random positions. At every instant of time the agent will choose one of his 8 possible actions, there's 20% of chance of picking a totally random action, although in 80% of the cases the agent will opt by the action that he considers the best to take in his current state, based on a table of states to actions. After executing a action the current state of the agent will be updated and the Q value for that action on the table will receive a reinforcement, that can be positive or negative, depends on the action.

The following actions generates a positive reinforcement:

- Attack an enemy.
- Reach the objective, while not adjacent to an enemy.
- Approach the objective, while not adjacent to an enemy.

The following actions generates a negative reinforcement:

- Attack empty tiles.
- Ignore the adjacent enemies by choosing any action other than attack them.
- Move away from the objective.

The reinforcement is passed to the gain function of the Q-Learning and the value of the action is updated. The cycle will be repeated many times, until that, eventually, all the weights will be adjusted, and the agent will be capable to trace a route from any position in the matrix, and beside that, he'll learn to not ignore is enemies, attacking they everytime they're adjacent, reaching a optimized state.

B. Victory and Defeat State

The agent and the enemy have health points value (HP), and attack points. The agent starts with 100 HP, and have 50 attack points, the enemies start with 100 HP and have 30 attack points. If the agent ever ignore an adjacent enemy, he'll suffer an attack from that enemy and his HP will be reduced. If his life reaches

zero, the agent will die. The agent will reach the defeated state everytime he dies or if he enter in an tile occupied by an enemy. The victory state is achieved when the agent reaches the objective tile, even if he didn't have defeated all the enemies. Achieve a victory or defeat state means that the instance will be restarted, all the enemies are respawned in the same tile, and the agent is respawned in a new random tile, but everything he learned will be kept for the next generation.

VIII. Results Evaluation

In order to observe the effectiveness of the algorithm, experiments were performed and log files were extracted. The analysis was performed in two different ways, and we will see each one separately.

C. Turn Point Valuation

Due to the unsupervised learning characteristics of the Q-Learning algorithm and the intrinsic randomness of the environment proposed in this project, it's necessary to adopt a policy to evaluate the results of intelligent agent training. Let's consider as a turning point the generation were the number of victories exceeds in 10 the number of defeats. The tests were performed in matrices of three different sizes, the results were extracted through log.

The agent was trained in two matrices of different sizes: the first a 5x5 matrix and the second a 10x10 matrix. The program was executed 10 times in each matrix, the results can be observed in figures 1 and 2.

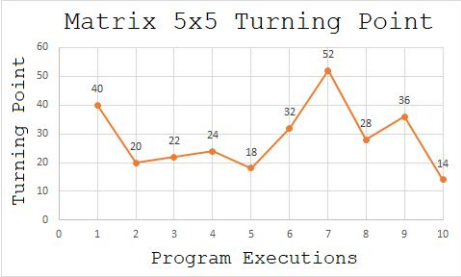

Figure 1. Observation of the Turn Point in a 5x5 Matrix.

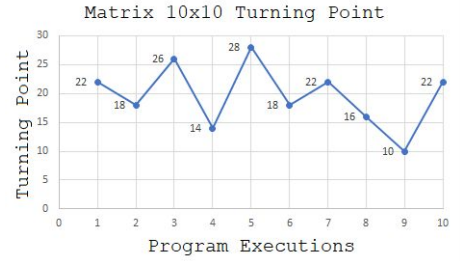

Figure 2. Observation of the Turn Point in a 10x10 Matrix.

From the analysis of the two graphs we noticed that in the 5x5 matrix the values of the turning point were much higher than the turning points of the 10x10 matrix. We can then conclude that since both matrices have the same number of enemies, it is easier for the agent to minimize errors in larger matrices, without so many enemies in its path, the environment becomes more permissive to route errors, causing the agent achieve victory status very easily.

D. Time-Lapse Evaluation.

Another way to observe the behavior of the agent is to analyze only one execution of the program over a large number of generations, allowing us to observe how long the agent takes to reach its maximum learning state. The agent was trained for 1000 generations in matrices with 3 different configurations.

```
2/1/2017 1:10:08 AM]: Agent   instantiated in tile [6][3]
2/1/2017 1:10:08 AM]: Goal instantiated in tile [3][5]
2/1/2017 1:10:08 AM]: Inimigo1 instantiated in tile [1][3]
2/1/2017 1:10:08 AM]: Inimigo2 instantiated in tile [5][3]
2/1/2017 1:10:08 AM]: Inimigo3 instantiated in tile [3][2]
2/1/2017 1:10:08 AM]: Generation: 10 victories: 6 losses: 4
2/1/2017 1:10:08 AM]: Generation: 20 victories: 13 losses: 7
2/1/2017 1:10:08 AM]: Generation: 30 victories: 21 losses: 9
2/1/2017 1:10:08 AM]: Generation: 40 victories: 31 losses: 9
2/1/2017 1:10:08 AM]: Generation: 50 victories: 40 losses: 1(
2/1/2017 1:10:08 AM]: Generation: 60 victories: 50 losses: 1(
2/1/2017 1:10:08 AM]: Generation: 70 victories: 60 losses: 1(
2/1/2017 1:10:08 AM]: Generation: 80 victories: 69 losses: 11
2/1/2017 1:10:08 AM]: Generation: 90 victories: 78 losses: 12
2/1/2017 1:10:08 AM]: Generation: 100 victories: 88 losses: 1
```

Figure 3. Example of a log file for Time-Lapse Evaluation.

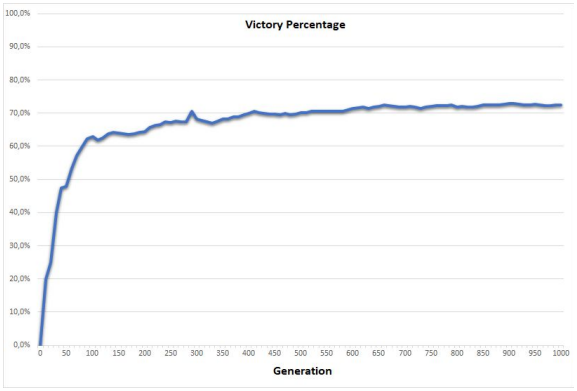

Figure 4. Example of a log file for Time-Lapse Evaluation.

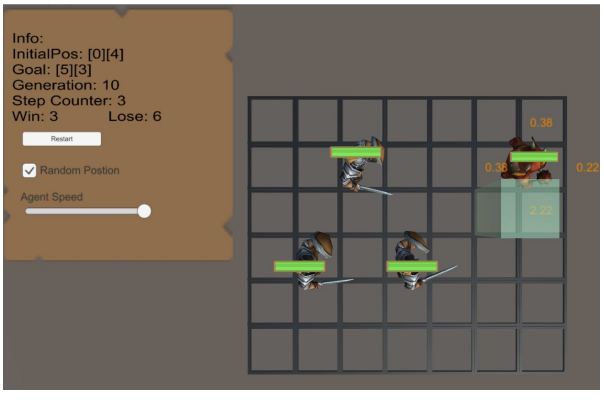

Figure 5. Configuration of the Matrix referring the Graph of Fig. 4.

The test represented by figures 5 and 6 occurred in a 7x6 matrix, with the objective in tile [5] [3], and the enemies in tiles [1] [2], [2] [4] and [3] [2]]. After 1000 generations the agent achieved a victory rate of 72.4%.

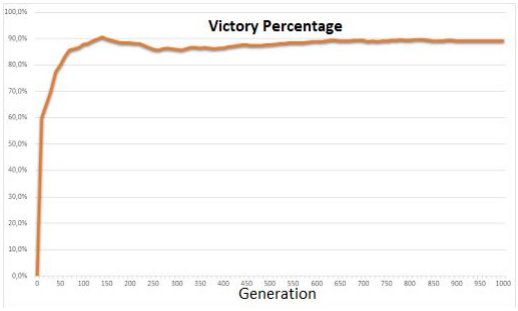

Figure 6. Graph of percentage of victories in a thousand generations 2.

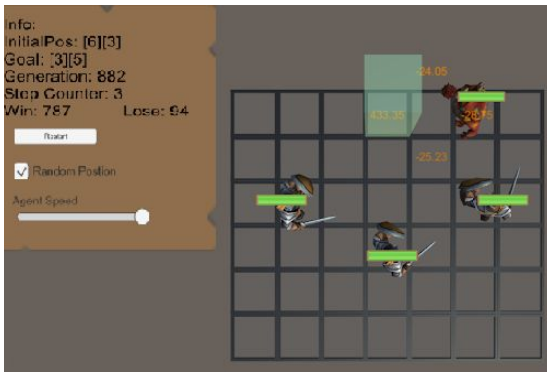

Figure 7. Configuration of the Matrix referring the Graph of Fig.6.

The test represented by figures 7 and 8 occurred in a 7x6 matrix, the objective being in tile [3] [5], and the enemies in tiles [1] [3], [5] [3] and [3] [2]. After 1000 generations the agent achieved a victory rate of 89.2%.

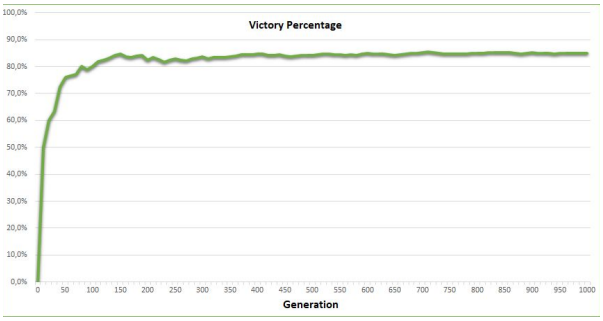

Figure 8. Graph of percentage of victories in a thousand generations 3.

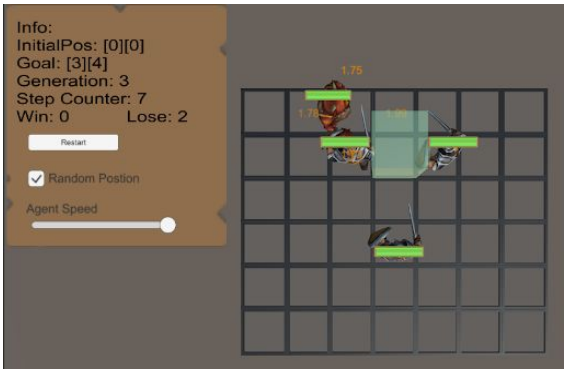

Figure 9. Configuration of the Matrix referring the Graph of Fig. 8.

The test represented by Figures 9 and 10 occurred in a 7x6 matrix, the objective being in tile [3] [4], and the enemies in tiles [2] [4], [4] [4] and [3] [2]. After 1000 generations the agent achieved a victory rate of 85.0%.

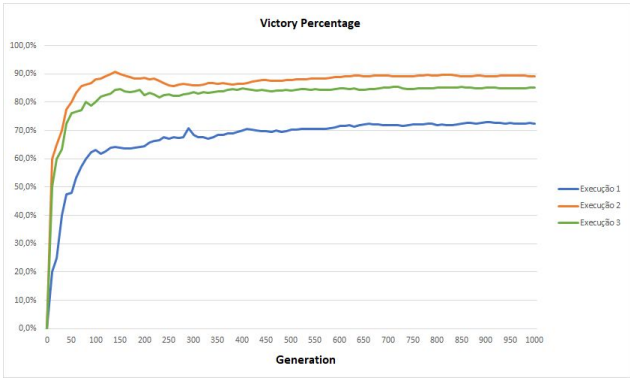

Figure 10. Comparison between the three executions.

After analyzing all the executions, we can see that in the first 100 generations, the agent reaches his maximum learning capacity, and that in about 400 generations the agent's winning rate changes very little, tending to remain stable at future time stamps.

We also verified that in the first execution the learning rate of the agent was much lower than in the other two, we can conclude that the fact that we have positions in the matrix in which the agent is adjacent to two enemies at the same time affects the good functioning of the learning process.

IX. Conclusion

In the field of Artificial Intelligence for Games, the Machine Learning Techniques have an enormous capacity of implementation to be explored, they have great uses isolatedly, and even more potential if we bind then together.

The test phases proved to be extremely important, since it made clear a failure in the agent's learning in situations where, incidentally, it would be adjacent to two enemies. Therefore, this failure should be addressed in future implementations.

We analyze the various steps of implementing an intelligent agent, and we can see how it behaved in different situations. The dynamism of Q-Learning has helped to produce an agent capable of generalizing and reacting to different environments.

References

[1] I. Millington and J. Funge, Artificial Intelligence for Games. CRC Press, 2006.

[2] R. Sutton and A. Barto, Reinforcement Learning: An Introduction. MIT Press, 1998.

[3] T. Mitchell, Machine learning. Boston: McGrawHill, 1997.