Solving Vermilion Gym Using Reinforced Learning

Aaron Cuadras
Department of Computer Science
University of Texas Rio Grande Valley
Edinburg, TX
aaron.cuadras01@utrgv.edu

Juan Martinez
Department of Computer Science
University of Texas Rio Grande Valley
Edinburg, TX
juan.martinez24@utrgv.edu

***Abstract*—Reinforced Learning is one of the most common practices in which we use Machine Learning to exploit the rewards and goals of any game. Puzzle games are the most popular for Reinforced Learning since some of them have a random nature and in other ones the player has to predict the movements of their opponent. Pokémon Red and Blue, the role-playing videogame released in 1996 for the Gameboy, includes a puzzle in which our agent will try to solve it in the fastest way possible using several Reinforced Learning techniques, and we will compare them to show the most adequate Reinforced Learning algorithm that solves this problem.**

I. Introduction

A. Motivation

We wanted to create a project in which we can benefit from Reinforced Learning to solve complicated puzzles, and this puzzle had the random nature that would make conventional methods for finding a solution fail. We also wanted to analyze the difference in performance of several reinforced learning algorithms to find the best fit to solve puzzle/random puzzle problems.

B. Previous Work

The work that has been done before is similar to the dynamic of moving in Pokemon, but instead of doing it with that game, the game used was Super Mario Bros. in which the machine tries to and the shortest path in the worst level and learns how to use the controls to play with the game. Since Pokemon is a game without a specific goal in mind it is really difficult to try to make the machine learn where does it have to go, so machine learning projects involving Pokemon have been focused on creating an optimal battle strategy depending on every Pokemon stat and try to predict which moves the other trainer will choose. Another difficulty with Pokemon is NPC's, a machine has difficulties trying to interact with them because they are a way of reinforcing the knowledge of the environment for humans instead of machines, they give tips to help the player move and explore the world and try to reach one of the goals of the game.

II. Problem

The Vermilion City Gym contains a switch-finding puzzle inside a 3 x 5 matrix of trashcans. The switch is located randomly in 8 of the 15 locations: in the first and third row, the switch could be located in the first, third or fifth column and in the second row the switch could be located in the second or fourth column. The objective is finding the hidden switch and once it is found, a second switch will be generated at the first switch side. If the player doesn't guess the second switch in the next check, both switches will be randomized, and the player has to start from 0. There is no way that the player can know where the first switch is located, but there is a strategy to find the second switch found in Fig.(*pending*), the only second switches that are placed in random in three places is when the first switch is located in trashcan [0, 2] and [2, 2]. The biggest problem for this project is finding the first switch since is the one randomly placed.

III. Agent and Environment

We implemented a simple bi-dimensional matrix (3 x 5) to imitate the trash cans in the gym, each time our agent moves to a location, it will check if the switch is there so moving to a location implies checking if the switch is there. Our agent has 4 states that it can change to simulating the movement of the player in -game (up, down, left, right). The agent cannot jump from one location to the other, so it does a systematic check in the path of trashcans, but we consider that a movement instead of a failed check.

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Fig. 1. Switch matrix, 1 represents that a switch can appear there

IV. Methods

A. Random Player

The first method implemented for this project was the Random Player Method. The agent will change from one state to another randomly, it will have an inconsistent behavior meaning that it will move around and can either take a long amount of time as it can take a small amount to find the switch, it will not remember the pattern or even follow one, but it is a solid start to make our agent travel through the environment. Our agent using the random player method takes between 0.02 to 0.08 seconds, four a human player is incredibly fast, but we can make it work faster and with a strategy.

B. Markov Chaining

Our agent will make a decision process named after the **Markov Property** which states that *The conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it.* Therefore, our agent will make use of a transition matrix to make the decision of changing states.

$$T = \begin{bmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{bmatrix}$$

Fig. 2. Transition Matrix for the agent

There is a 70% probability of changing to the same state and a 30% to any other state. This makes our agent to follow a trend, but still make the performance of it really inconsistent because it will keep hitting the walls for several turns until it changes to another state that is not the same one, performance for this agent ranges from 0.02 to 0.04 seconds and our performance graphs shows an inconsistency between one episode and the next one with sudden spikes meaning that the agent is hitting the wall constantly.

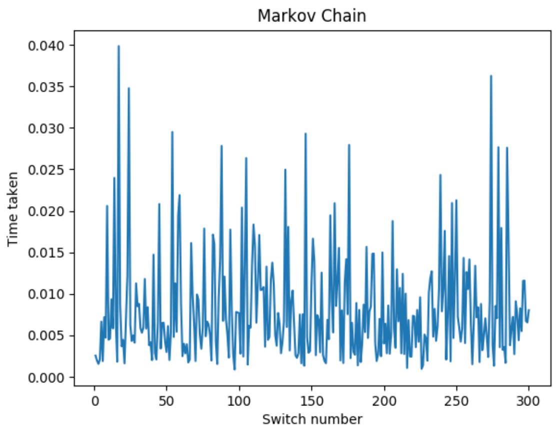

Fig. 3. Performance for Random Player (Time vs. Switch Found)

C. Q-Learning

Reward based takes into consideration past choices and the outcome of those choices. The agent interacts with the environment in one of two ways: Exploration (more random) and Exploitation (greedy, always seeks highest reward). Both exploration and exploitation can be tuned to achieve different results. If there is low exploitation value, the algorithm will only choose paths with a high rewards (greedy). On the other hand, if the exploitation value is high, the algorithm will explore other paths to solve the puzzle.

$$\widehat{Q}(s,a) = R(s,a,s') + \gamma max_{a'} Q_k(s',a') \tag{1}$$

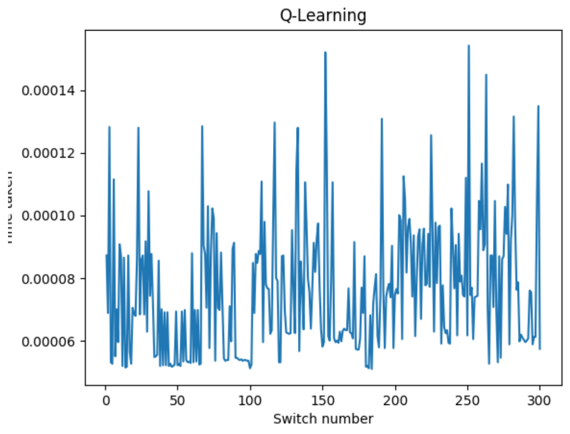

Fig. 4. Performance for Q-Learning (High Gamma Decay)

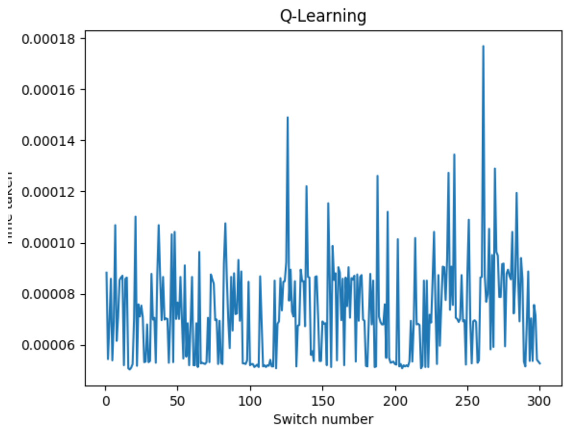

Fig. 5. Performance for Q-Learning (Low Gamma Decay)

V. Conclusion

This projects had the goal of finding a fast algorithm which finds a switch fast and in a consistent time frame, our agent learned more effectively with Q-Learning using a low *decay gamma*, this means that our agent will prioritize exploiting the environment instead of exploring other paths to find an optimal solution. Since our environment is so small exploring it means taking more time to find options in which the agent will stay far away from the path instead of exploring without straying away from the path. We considered that our three methods were a progressive measure in which we construct an agent to start exploring an unknown environment, the **random player** method is an example on how an agent will try to move in an unknown environment, the **Markov's chain** method is an example on how the agent starts to follow a certain trend to make a more consistent movement in the environment and **Q-Learning** is a method in which our agent starts learning which one is the reward and try to find the optimal solution to the puzzle.

VI. Acknowledgment

We want to thank Dr. Dongchul Kim for lecturing us in Machine Learning and by giving us insight for our project, as well to our peers for giving us the tools to improve our project and for suggesting on how to make a better agent.

References

[1] Shayaan Jagtap, "Why Can a Machine Beat Mario but not Pokemon?," https://towardsdatascience.com/why-can-a-machine-beat-mario-but-not-pokemon-ff61313187e1

[2] Devin Soni, "Introduction to Markov Chains," https://towardsdatascience.com/introduction-to-markov-chains-50da3645a50d

[3] Andre Violante, "Simple Reinforcement Learning: Q-learning," https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[4] JaeDukSeo, "Reinforcement Learning an Introduction," https://github.com/JaeDukSeo/reinforcement-learning-an-introduction

[5] David Silver, "UCL Course on RL," http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html