The Game of Pac-man in AI

The Purple Potatoes:
Steven Vo
Aaron Grodzinsky
Marie Blurton-Jones
Lucy Xia

***Abstract*—PAC-MAN in AI is a program that is essentially the game PAC-MAN. The only different here is that the character, Pac-man, will be an Ai agent. Pac-man will learn how to move around the maze, collect Pac-Dots, and dodge ghosts through reinforced learning.**

I. Introduction

This project will be based off on the game Pac-Man. Pac-Man is a character, which a player controls, that moves around a maze collecting dots known as Pac-dots. While collecting these dots, the player must also avoid ghosts that move around the maze. What this project is trying to accomplish is making Pac-Man an Artificial Intelligence Agent. This means that Pac-Man will not be controlled by a player and will make decisions on its own based on where Pac-dots and ghosts are through the reinforced learning algorithm, Q Learning.

II. Motivation

Our motivation for this project stemmed from our desire to better develop our problem solving skills and our ability to apply the artificial intelligence theories we have learned about in class to a real world example. We wanted to take a well known fun to play game, such as Pacman, and figure out the best way to enhance it using artificial intelligence. We believe that we can enhance the user experience of the game of Pacman and increase the users desire to play the game more through the implementation OF AI. By implementing the algorithms and python libraries we have found in our research of this task, we will be able to achieve this goal.

III. Data Sources

While developing this Pac-Man AI, there are going to be a numerous amount of data sources that we are going to keep track of to provide analyses on how efficient our agent is. At the start, we will record the path of Pac-Man while it tries to learn how to navigate around the maze. We will also record the time it takes to collect each Pac-Dot. We will do this three times. Once when the agent has randomized movement, again when the agent moves based on a simple searching algorithm, and lastly when the agent is implementing the Q learning algorithm. This will demonstrate that our agent is able to make precise, accurate, non random decisions based on the environment. Also, we will record if our Pac-Man agent completes or fails in getting all the Pac-Dots before it is hit by a ghost. It may be useful to see if there is a pattern to how exactly Pac-Man is failing at accomplishing its task.

IV. Algorithms

In this game of Pac-man, we would be primarily using reinforcement learning and supervised learning techniques - subsets of machine learning in which the computer learns from experience. In this case, there would be two agents - Pac-man and the ghosts. These agents will exist within an environment of a grid, which will be constructed as a 2d array of objects backed by list of lists. The data is accessed via grid[x][y] where (x,y) are positions on a Pac-man map with x as the horizontal coordinates, y as vertical coordinates, and the top left corner - (0,0) as the origin.

A. More on reinforced learning

Reinforce learning is where an agents takes in information on the environment and decides what is that best action to take based on the information it got. In other words, the agent's action will affect the state of the environment, in this case is the maze, and what reward is given to the agent.

Initially in our project, the agents will be given many test-cases and solutions from various search algorithms. The search algorithms will be designed to iterate through and check sections of the environment to see which spaces have been previously visited, if going forward will be an invalid move that violates the grids boundaries, and create solutions to possible scenarios with the ghost agent acting as a reflex agent. The end goal is for our

pacman agent to be able to define an action or movement in which it determines whether or not to move North, South, East, West or come to a stop based on the Q Learning Algorithm. In our project, we will be implementing PyBrains Q Learning algorithm to maximize the accuracy in the algorithm, which translates to our agents movements[5].

B. Q Learning Algorithm

The Q Learning Algorithm is a type of function in the form of reinforcement learning that learns the most efficient state-action pairs, or the most efficient action to take in the current state. It provides steps for the agent to be capable of learning how to act optimally in various domains with a sequence of actions. The procedural approach to this algorithm is summarized as follows:

1) Initializing the Q-values table, or the domain
2) Observing the current state
3) Choosing an action
4) Taking the action and observing the reward, as well as the new state
5) Updating the Q-value for the state using the observed reward and the maximum reward possible for the next state through a specified formula
6) Updating the state into a newer state
7) Repeating the process until terminal

When applying this learning algorithm to Pacman in AI, we see that this concept illustrates the way the agents in our game (Pac-man) traverses the domain (maze), collect the Pac-Dots as rewards, and receive the maximum possible reward. This reward will be calculated based on the agents current position, its current goal - Pac-Dot, the length of the optimal path, and the proximity to the ghost reflex agents.

V. Related work/literature review

"Teaching introductory artificial intelligence with pac-man" gives a brief description on a multitude of learning techniques that can be implemented to effectively have both Pac-Man and the ghosts run as AI agents. The first two implementations used breadth-first and depth-first search algorithms to navigate through the Pac-Man maze allowing the agent to find the next Pac-Dot to move to. This seemed to be a good starting point for their project, but it was very simple. It did not provide any form of recognizing how to effectively escape from the ghosts and it did not optimize the agents path to the Pac-Dot.

To combat this, the team implemented reinforcement learning techniques, such as Q Learning, with a ghost adversary acting as a reflex agent. This, along with the knowledge we gained during our lecture, motivated us to implement the Q Learning algorithm for our AI agent. In a reinforcement learning problem, Q Learning is a perfect solution. It allows an agent to maximize the expected total reward that would be gained from possible steps, starting from the current state. This not only allowed the team to have a simple design for the Pac-Man game to run fully through AI, it also allowed the team to observe the behaviors that emerged from the two AI trying to beat one another. As development of our project continues, we will have to watch how our AI agents learn to escape, or chase each other in an effective manner.

Another algorithm that we could have implemented for Pac-Man is an All-Moves-As-First (AMAF) algorithm. A study done at the University of Beijing by Bin Wu showed the success rate of this algorithm compared to a Upper Confidence Bound for Trees (UTC) AI algorithm. (Wu) This implementation used AMAF algorithm, which the report mentions is similar to the AMAF algorithm used in the game of Go, just with the addition of children nodes. The children nodes represent the directions in which the player can move. The implementation also attempted to improve efficiency by implementing a rule-based policy. This was because the rule-based policy was able to improve efficiency by saving the simulation time, and partly because that the moves with highest AMAF value are sometimes not the globally better moves. (Wu) The implementation included running the game 300 times with the rules in place. The outcome was that the AMAF implementation worked well as a single player game, and as time progressed its performance came close to the UTC algorithm implementation, but that in a multiplayer online game version of pac-man the AMAF implementation would prove insufficient. (Wu)

Recent developments in the Pac-Man universe also include the recent implementation of the game of Ms. Pac-Man by Maluuba, a Canadian deep learning startup that was recently acquired by Microsoft. Maluuba was able to use reinforcement learning to play Ms.Pac-Man perfectly, obtaining the maximum score of 999,990. (Linn) Instead of implementing just one AI agent, the team implemented over 150 agents, calling it Hybrid Reward Architecture. Each agent worked in parallel to master the game. For instance some agents were rewarded for avoiding ghosts, while others were tasked and rewarded with finding a specific pellet. The player

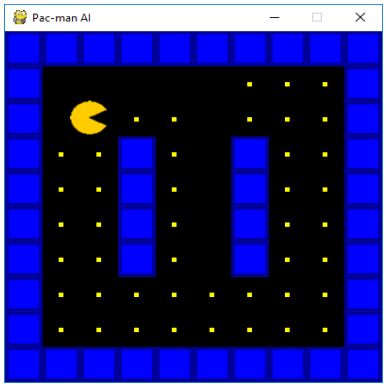

Fig. 1. The Pac-man AI agent traversing through the maze using its searching algorithm.

Trial Number	Completion Time
Trial 1	8 min. 35 sec.
Trial 2	6 min. 54 sec.
Trial 3	15 min. 13 sec.
Trial 4	12 min. 49 sec.
Trial 5	5 min. 43 sec.
Average Time	9 min. 59 sec.

Fig. 2. This table holds the data for the time it took the Pac-man AI agent to collect all of the Pac-Dots with randomized movements.

was ultimately controlled by the top agent, who took suggestions from all the other agents, and the intensity of their suggestions, into account before making a move. (Linn) For example, the top agent would give more weight to the sub-agents that were saying to move in a specific direction if there was a ghost in the other direction. Unlike in our implementation of Pac-Man, this implementation of Ms.Pac-Man used reinforcement learning to train the AI. AI experts believe that reinforcement learning could be used to create more AI agents that can make more decisions on their own, this would allow them to do more complex work and free up people for more high-value work. (Linn)

Trial Number	Completion Time
Trial 1	52 sec.
Trial 2	55 sec.
Trial 3	56 sec.
Trial 4	51 sec.
Trial 5	49 sec.
Average Time	53 sec.

Fig. 3. This table holds the data for the time it took the Pac-man AI agent to collect all of the Pac-Dots with randomized movements.

VI. Preliminary Results

To portray the effectiveness of implementing the Q Learning Algorithm, we programmed the agent to collect all the Pac-dots in a simple 10 by 10 environment with two wall obstacles and 56 Pac-Dots. This removed the ghost reflex agents from the environment to test our Pac-man agent's efficiency with collecting the Pac-dots. We programmed the agent to traverse the environment using Random Movement, Algorithmic Movement, and Q Learning Movement to compare different techniques of creating a Pacman Agent.

1) Randomized Movement: First, the agent traversed the environment with random movement. This allowed us to see how the Pac-man agent would interact with the environment and Pac-Dots in the most simplest form possible. In addition, it demonstrated the need and use for implementation of the Q Learning Algorithm. We took 5 tests to generate an average completion time to compare with the other movement techniques. As you can see in Fig 2, the randomized movement was very inefficient and inconsistent. The average time of completion is 9 minutes and 59 secs.Throughout our tests, the agent would frequently get stuck in random areas when the remaining Pac-Dots were on the opposite side of the environment.

2) Algorithmic Movement: The next movement technique revolves around an algorithm that finds the Pac-Dot closest to the Pac-Man agent and generates a direction and initial path for the agent to follow. The closest Pac-Dot was determined by the distance formula:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

The algorithm successfully performed its task. Something that was not considered was that this algorithm prioritized what direction the Pac-man agent went. It prioritized going up, left, right, and then down. In Fig 3, the results were far better than Fig 2. The average time of completion for the algorithmic movement was 53 seconds, which is much less than the randomized movement average.

3) Q Learning Movement: The most efficient movement technique is the Q Learning Movement. This style of movement implements the Q Learning Algorithm as described above. At first, the Q-Learning time until collection of every Pac-Dot resembled the time of Random

Movement's collection, maxing at 15 minutes and 33 seconds. However, as more and more attempts were presented to the agent, it slowly learned the best path to receive the maximum rewards. After 100 tests, the Pac-Man agent was collecting all the Pac-Dots in the environment in 56 seconds.

VII. Environment Test Data

We tested our Pac-Man agent in several different environments ranging from very simple maze to a representation of a standard Pac-Man game, implementing our three movement algorithms to compare and contrast the effectiveness of the Q-Learning algorithm.

A. Movement Test Environment

This environment was used to test the various movement algorithms implemented in our Pac-Man program. It allowed us to compare and contrast the algorithms in a basic maze environment. The environment is a simple 10x10 grid with two wall columns in the middle to provide boundaries. In this environment the spawn point of Pac-Man was also randomized in order to further test the variability of our algorithms.

In this environment, our results were based on 5 tests, excluding Q-Learning which underwent 100 tests. The data for this section can be found in figure 6. In our results, the randomized movement time ended up taking an average of 9 minutes and 59 seconds (Fig. 2). Our Breadth-First Search result, which was also based on 5 tests, resulted in an average of 53 seconds (Fig. 3). After 100 tests our Q-learning algorithm produced a result of 56 seconds. Based on the results seen in figure 4, the Q-Learning Algorithm proved to have steadily decreased time until completion of task as it progressed through each test demonstrating the learning process.

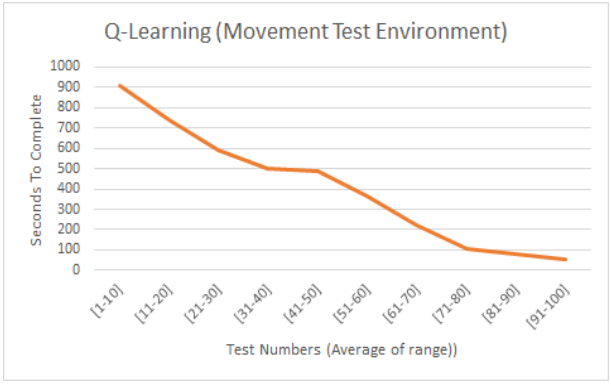

Fig. 4. The Pac-man AI Movement Test Environment Data

B. Ghost Evasion Environment

The next environment that was tested was the Ghost Evasion Environment. This environment, pictured in figure 5, was implemented to test our agent's ability to avoid the ghost reflex agents while still collecting Pac-Dots. We introduced our ghost reflex agents to the maze, locking them in a vertical up and down movement to provide a simple standard test for our agent to interact with. This maze was 10x7, without any barriers inside, with one Pac-Dot located at location (8,3) in the environment on the side opposing the agent's spawn location, (1,3).

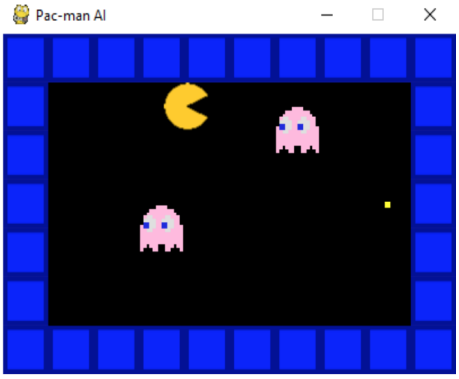

Fig. 5. The Pac-man AI Ghost Evasion Environment

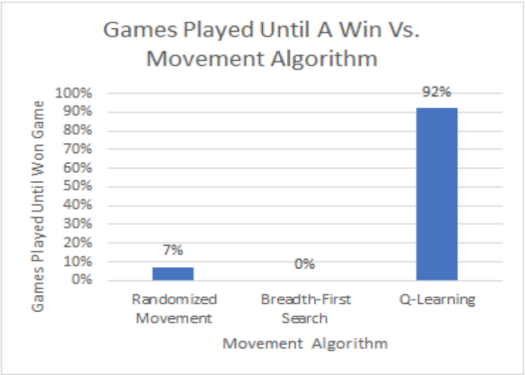

Fig. 6. The Pac-man AI Ghost Evasion Environment Data

In the Ghost Evasion environment our data results were based on 100 tests for each movement algorithm. We calculated win percentages, with a win characterized as the agent collecting the only Pac-Dot in the environment. These percentages can be seen in figure 6. For Randomized Movement, only 7% of the tests completed had a win outcome. This percentage is low due to the movement being completely random, so the agent collecting the only pac-dot is all from luck. The result of the Breadth-First Search algorithm was even

lower at 0%. The reason why the percentage is 0% in this case is because our Breadth-First Search algorithm did not account for the ghost reflex agents. The algorithm would simply tell our Pac-Man agent to move in the direction of the dot ignoring its consistent overlap of the ghost. In contrast, Q-Learning Algorithm produced a win percentage of 92% after approximately 70 tests of the 100 tests performed in the Ghost Evasion Environment. Just like in the Movement Test Environment, the Q-learning process started off sporadic, similar to the random movement. However, over time our agent learned to evade the ghost and collect the dot consistently due to receiving a higher reward.

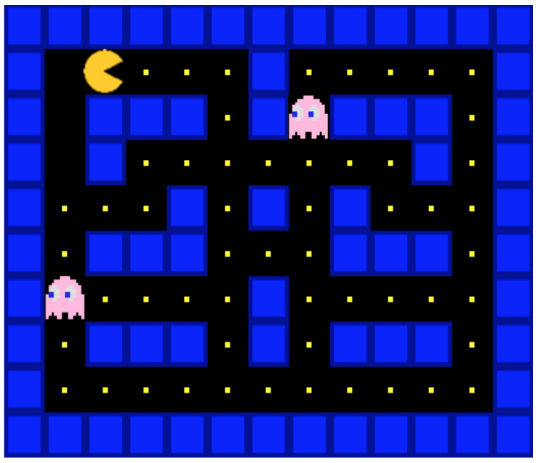

Fig. 7. The Pac-man AI Standard Small Map Environment

C. Standard Small Map

The third environment that we tested was a standard small map environment. Looking at figure 7, it is a 13x10 maze with several barriers throughout to provide many obstacles for the agent to learn. The environment contained two ghosts, one Pac-Man agent, and 61 Pac-Dots. This environment is the first of two environments that we tested that most closely resembled the standard Pac-Man map. Pac-Man is always spawned in the top left corner at location (1,1). This was done to increase the learning speed for the Q-learning agent. The task of the agent was to collect all of the Pac-Dots while effectively evading the ghosts.

In the Random Movement Algorithm tests for our Standard Small Map environment, there were very sporadic outcomes. It took a total of 233 games for the random movement algorithm to collect all 61 Pac-Dots while managing to avoid the ghosts. This test was driven by luck due to it being randomized. The results of each of the test are shown in figure 8.

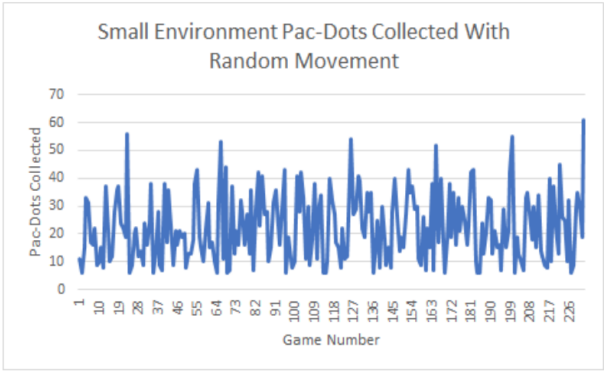

Fig. 8. The Pac-man AI Standard Small Map Random Movement Data

The results of the Breadth-First Search Algorithm implementation in the Standard Small Map took far longer than expected. This was due to the algorithm ignoring the position of the ghosts in its search. It took 127 games for the Breadth-First Search Algorithm to collect all 61 Pac-Dots in one game. The data from the Breadth-First Search Algorithm can be found in figure 9.

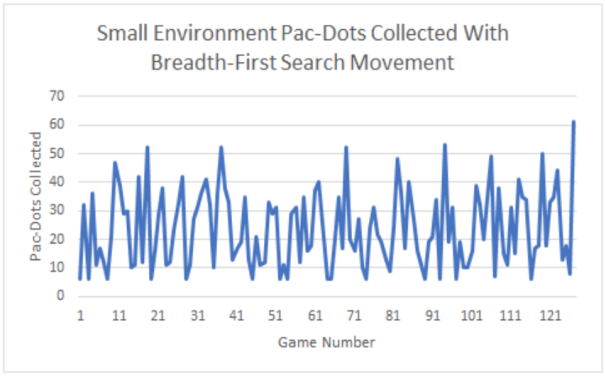

Fig. 9. The Pac-man AI Standard Small Map Breadth-First Search Data

The Q-Learning Algorithm was tasked in this Standard Small Map environment to learn how to traverse the map and collect Pac-Dots while dodging ghosts through reinforced learning. It only took 91 games for the Q-Learning agent to collect all 61 dots in one test. One complication we ran into was the average the length of each of these test, which was approximately 9 minutes during the later stages of testing. Figure 10 shows the data for the Q-Learning algorithm in the Standard Small Map environment.

As seen in the figures mentioned above, both random movement and breadth-first search movements graphs

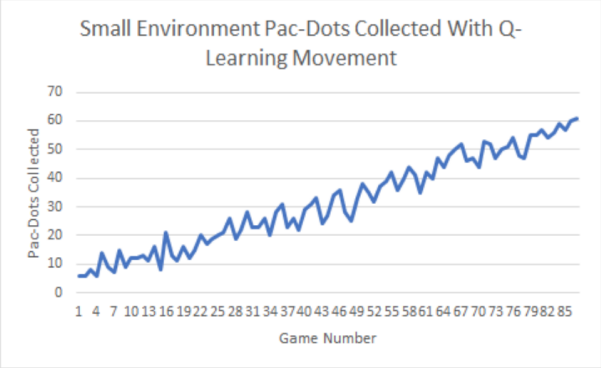

Fig. 10. The Pac-man AI Standard Small Map Q-Learning Data

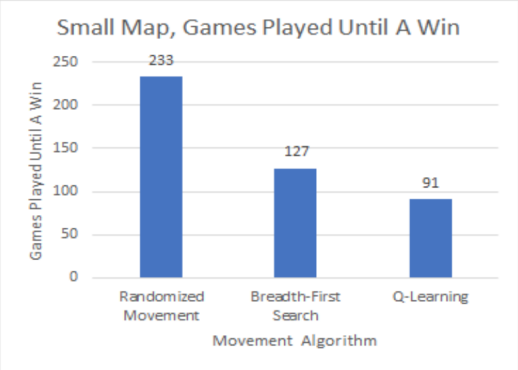

Fig. 11. The Pac-man AI Standard Small Map Algorithm Comparison

were very sporadic. This is due to the variability caused by both movement and the ghosts. The Q-Learning graph however showed a steady yet still slightly inconsistent growth to eventually reach a win in a shorter amount of games. In figure 11, we compare the number of games each movement algorithm played until they successfully collected all the Pac-Dots. Unsurprisingly, Random Movement required the most games, while Breadth-First Search and Q-Learning were close in their games needed to complete this task. We hypothesize that this is because this map has a small number of states and possible moves, causing both algorithms to not require many trials to succeed.

D. Standard Large Map

To test our hypothesis, we created one more environment, the Standard Large Map environment. This environment is the second of the two that resemble a standard Pac-Man map involving ghosts and Pac-Dots. This is the largest environment with the size if 15x15. The environment contained Pac-Man, 4 Ghosts and 115 Pac-Dots. Due to the large number of possible states, the Agent learned to interact with this environment very slowly. Similar to the smaller map, Pacman was always spawned at location (1,1). This environment can be viewed in figure 12.

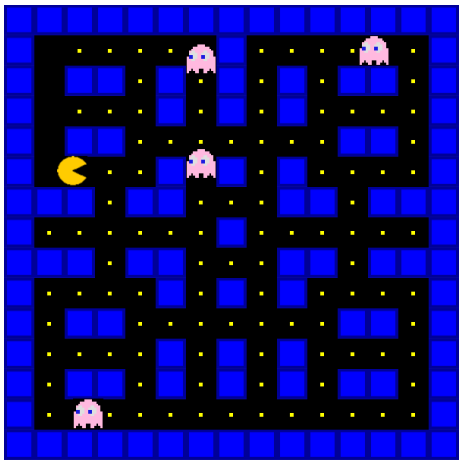

Fig. 12. The Pac-man AI Standard Large Map Environment

The implementation of the Random Movement algorithm in the Large Standard Map environment took a very long time to get a successful win. This was due to the agent's random movements since the agent is ignoring how close each ghost is and where the cloest Pac-Dot is. It took 833 games for this algorithm to collect all 115 Pac-Dots while simultaneously avoiding all the ghosts. The data for this algorithm can be seen in figure 13. This test was driven by luck because it is all randomized causing this absurdly large number.

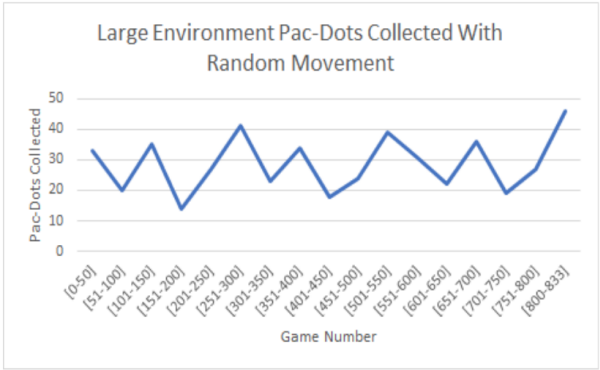

Fig. 13. The Pac-man AI Large Standard Map Random Movement Data

The agent also took a long time to complete its task while implementing the Breadth-First Search movement in this environment. It took 629 games for our agent to collect all 115 dots using this algorithm. The data for this algorithm can be seen in figure 14. Like the Random

Movement, this test is essentially driven by luck based on the random movements of the ghosts. The ghosts spawn in the center square every single time, so if either of the ones that spawn to the left move up immediately, the agent will die within collecting 6 dots. However, if this does not occur, the agent frequently collects a larger number of pac-dots.

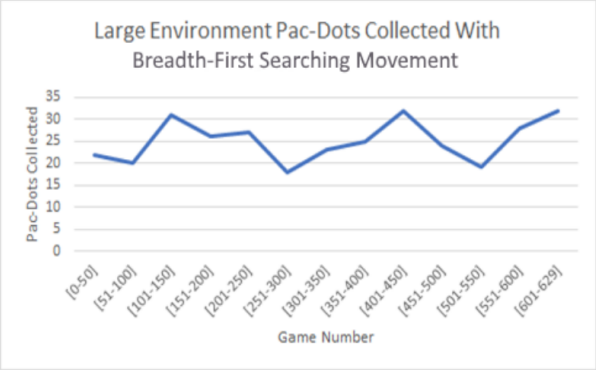

Fig. 14. The Pac-man AI Large Standard Map Breadth-first Search Data

The Q-Learning Algorithm only took 362 games for the agent to collect all 115 Pac-Dots while managing to evade the ghosts. This was the lowest amount of games needed to complete the task out of any of the movement algorithms. However, just like in the Standard Small map, the time it took for Pac-Man to play a game rose as the number of tests completed increased, tests taking an average of 25 minutes in the later phases. This caused the Q-Learning Algorithm to learn the task the slowest in reference to real time, but quickest in reference to games played. The data for the Q-Learning Algorithm movement can be seen in figure 15.

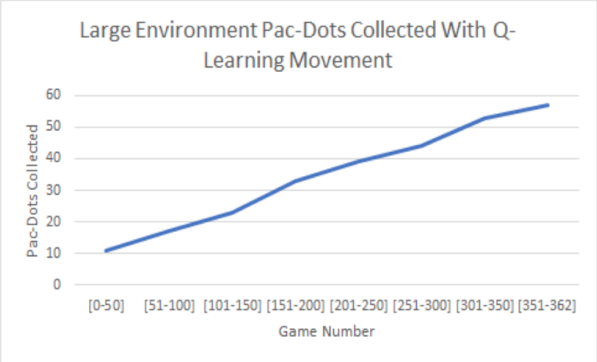

Fig. 15. The Pac-man AI Large Standard Map Q-Learning Data

Just like the Standard Small Map, both random movement and breadth-first search movements graphs were very sporadic. This is once again due to the variability caused by both movement and the ghosts. The Q-Learning graph however showed a very slow yet steady inconsistent growth to eventually reach a win. In figure 16, the games required until a win for each movement algorithm is compared once more. It can be seen now that there are more states, ghosts, and Pac-Dots to collect, the distance between Breadth-First Search and Q-Learning has increased.

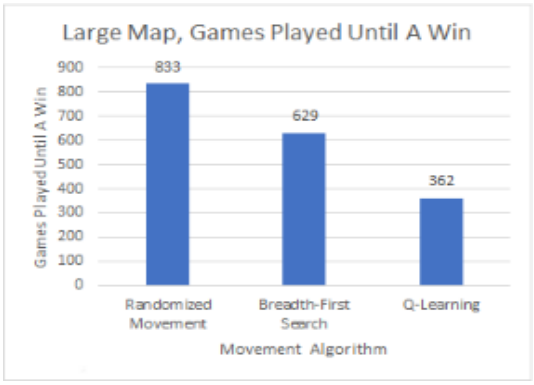

Fig. 16. The Pac-man AI Standard Small Map Algorithm Comparison

VIII. Comparing the Data of the Standard Map Environments

Comparing the data results from the tests of both the Standard Small Map Environment and the Large Standard Map Environment (shown in Fig. 11 and Fig. 16), it can be seen that Q-Learning is the fastest of the three movement algorithms as the number of states and ghost reflex agents increase. However, Q-Learning took the largest amount of real time to complete its tasks. This is an intriguing observation, and is most likely caused by imperfections in our Q-Learning agent.

In addition, through this comparison we can confirm our earlier hypothesis, that the number of states is proportional to the number of tests needed for a Q-Learning Agent to complete a task. For example, in an environment with 130 states and 2 ghosts, Q-Learning took 91 Games to learn. This is a small number compared to the 362 games it took for the agent to learn in the larger environment of 225 states and 4 ghosts.

IX. Conclusion

Given enough tests for the Q-Learning algorithm to learn from, our Pac-Man agent will be more successful with the Q-Learning algorithm than with the Breadth-First Search or the Random Movement Algorithm. However, in a simple maze solving program, a breadth-first

search AI might be more efficient than using Q-Learning. We also found that the larger the environment, the more learning trials needed for the Pac-Man Agent to learn how to correctly win.

REFERENCES

[1] DeNero, John, and Dan Klein. "Teaching introductory artificial intelligence with pac-man". *Proceedings of the Symposium on Educational Advances in Artificial Intelligence.* 2010.

[2] Linn, Allison. Divide and Conquer: How Microsoft Researchers Used AI to Master Ms. Pac-Man. The AI Blog, Microsoft, 14 May 2018.

[3] Wu B. (2012) The Computational Intelligence of the Game Pac-Man. In: Wang Y., Zhang X. (eds) Internet of Things. Communications in Computer and Information Science, vol 312. Springer, Berlin, Heidelberg

[4] Reinforcement Learning. Reinforcement Learning - Algorithms, www.cse.unsw.edu.au/ cs9417ml/RL1/algorithms.html.

[5] Tom Schaul, Justin Bayer, Daan Wierstra, Sun Yi, Martin Felder, Frank Sehnke, Thomas Rckstie, Jrgen Schmidhuber. PyBrain. To appear in: Journal of Machine Learning Research, 2010.