Final Report: Pac-Man Reinforcement Learning

Andrew Chen, Dhruv Aggarwal, Henrique da Ponte

Foreword:

This paper represents a much improved version of our project in comparison to the one we presented in-class. We took feedback and dug into what we knew and how we could improve our approach. Thus our conclusions are more evidenced and our model runs quite a bit better.

I. Description of Problem

Our group wanted to tackle the game Pac-Man using a reinforcement learning approach. The game itself is quite simple, with the simple objective being collect all the pellets and avoid touching the Ghosts. There are four directions Pac-Man can move and the environment lends itself to a GridWorld-like world with squares that Pac-Man can move to and obstacles that block his movement. What sets this apart is the addition of the Ghosts. The Ghosts move around the environment in either random or programmed behavior (such as directionally towards Pac-Man). This makes the state space exponentially larger as we would have to account for every possible permutation of the Ghosts' positions and pellet locations. Later we will describe our attempt to mitigate this by adding Q-value approximation. Our attempts include different Ghost strategies, such as random movement or chasing Pac-Man intentionally.

II. Models and Layouts

One of the challenges we wanted to give Pac-Man was to see his performance with a variety of maps and layouts. To start, we used the '*smallGrid*' layout as that was the simplest. We played with more complicated layouts as well, such as the '*mediumGrid*', '*smallClassic*', '*mediumClassic*', and '*trickyClassic*'. In this paper we will only discuss '*smallGrid*' and '*mediumClassic*' as those are the layouts we used the most.

fig 1, 'smallGrid' layout

fig 2, 'mediumGrid' layout

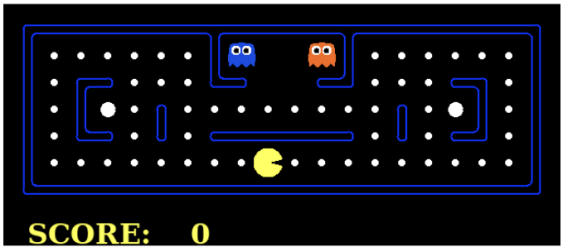

fig 3, 'smallClassic' layout

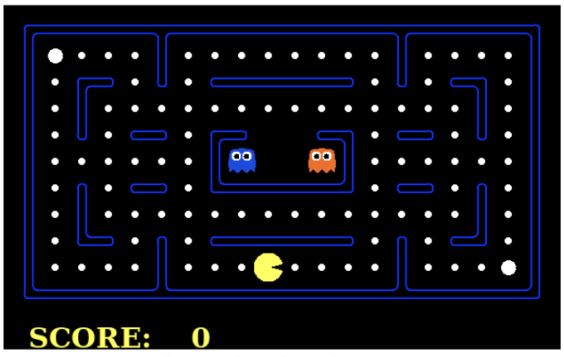

fig 4, 'mediumClassic' layout

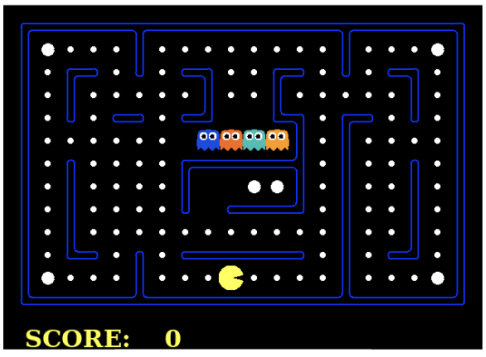

fig 5, 'trickyClassic' layout

III. Agents & Rewards

A. Pac-Man Agents

- Q-Learning Agent
  - Linear Q-Value Approximation

B. Ghost Agents

- Random Agent
- *'DirectionalGhost'* Agent

C. Reward Structure

- + 500 for winning the game (all food pellets are eaten)
- + 200 for eating a scared Ghost.
- + 10 for each food pellet eaten.
- -1 for each timestep (encourages the agent to finish the game quickly)
- -500 if Pac-Man gets eaten

IV. Initial Ghost Agent Structure (Random)

We decided to begin by creating an agent for how we wanted the Ghosts to operate and our first approach was a randomized agent, which would simply create a normalized distribution of all legal moves and then decide one at each timestep. We then moved on to creating Pac-Man's agent and did so using Q-Learning techniques.

V. Pac-Man Q-Learning Agent

We began by testing Pac-Man in small environments, such as the '*smallGrid'* layout shown in Figure 1. The state space for this was simply the location of Pac-Man and the Ghost, as the pellet was fixed to only two locations. This method involved storing the Q-value for every state within a Q-table, and then choosing the action with the best score after. However, we quickly found that this was ineffective with larger models.

VI. Additional Approach (Q-Value Approximation)

Once we began training on larger layouts such as the *'smallClassic', 'mediumClassic',* and even *'trickyClassic,'* we noticed that it was simply not feasible to train using Q-Learning. As the number of pellets and grid spaces increased, the number of possible permutations increased exponentially. This was not helped as we added in more Ghosts either. It is infeasible to store every Q-value for every combination of states; thus, we added a linear Q-value approximation technique that extracts a combination of features from the state-action pair.

Let S denote the state space and A the action space. For any state-action pair $(s, a) \in S \times A$*, we can define a feature vector* $\phi(s, a) \in R^n$*, where n is the number of features.*

The Q-value function $Q: S \times A \rightarrow R$ *is then approximated by a linear combination of these features:*

$$Q(s, a) = \sum_{i=1}^{n} \theta_i * \phi_i(s, a)$$

Where:

$\theta \in R^n$*: vector of weights*

$\phi(s, a)$: *feature vector for state-action pair (s,a)*

This allows us to approximate Q-values from feature weights to train our Pac-Man agent much faster. Some of the weights we used were:

[#-of-ghosts-1-step-away, closest-ghost, closest-food, closest-capsule, num-capsules-left, eats-food, closest-scared-ghost, eats-scared-ghost, missed-scared-ghost, current-score]

VII. Q-Value Approximation Results

With our Q-value approximation, we were now able to simulate the larger layouts without much trouble. While with the simple Pac-Man Q-Agent we were only able to train it on '*smallGrid*' successfully, we were now able to train our agent (which is referred to as our Approximate Q-Agent from now on) with the '*smallClassic*', '*mediumClassic*', and '*trickyClassic*' layouts which pose to be much more difficult than the '*smallGrid*' layout (as well as having a much larger state space). While our previous Pac-Man Q-Agent was not able to train well at all on the '*Classic*' variants of the layouts, our Approximate Q-Agent was able to significantly decrease the number of training episodes needed for a functioning agent.

On the '*mediumClassic*' layout and with *'RandomGhost'* behavior, we found that 50 training episodes was sufficient for Pac-Man to play and win most of the time (with *'RandomGhosts'* we found that to be the case with most of the layouts we played with). Training for longer episodes allowed the agent to optimize the '*current-score*' feature and increased the average score (during training). We did a total of 1000 training episodes and saw that the optimization seemed to stop at around 200 episodes, where we assume the maximum score from the game cannot be significantly increased any more. The learned behavior in this format seemed to be to just focus on eating the food and almost ignore the Ghosts unless they were 1-step away. The weights seemed to also showcase this behavior as the feature '*#-of-ghosts-1-step-away*' was highly negatively weighted while '*closest-ghost*' had a small (but negative) weight attached to it. Touching on the power-up capsules, Pac-Man doesn't appear to be highly desiring to go to a power-up capsule specifically, but when it eats one while playing the game, Pac-Man tries its best to eat all the Ghosts while they are scared. We can see the '*eats-scared-ghost*' feature only increase during training. At this point, Pac-Man won games almost every time with only small instances of accidentally getting caught between two Ghosts.

VIII. Directional Ghost

With Pac-Man succeeding at a near 80~90% win rate against the '*RandomGhosts*', we decided to implement another Ghost agent to challenge him further. This new '*DirectionalGhost*' prioritized chasing after Pac-Man by going in the direction that would minimize the Manhattan distance between them. On the contrary, when Pac-Man ate a power-up capsule, the Ghosts became afraid and would choose the direction that maximized this distance. This '*DirectionalGhost*' behavior was much harder for Pac-Man to learn and win against than the '*RandomGhost*' agents, winning a mere 25% of the time at best.

When we introduced '*DirectionalGhosts*', Pac-Man's behavior changed dramatically. Pac-Man's performance after and during training were both not as great as when the Ghost had random behavior but Pac-Man seemed to have a much more solid game plan. We found this version of Pac-Man to be a

much more patient player, willing to wait in one spot until the Ghosts had moved to a more favorable position before heading out to collect more pellets. What we saw was that Pac-Man would wait until the Ghost(s) were close such that it could lead the Ghosts around behind it so he wouldn't be facing the Ghosts head on; which we can gather would be much more difficult to navigate than just running away from the Ghosts. Interestingly we can see this by the '*closest-ghost*' feature which was *positively* weighted as opposed to previously where it was small but still negatively weighted.

Some other interesting behavior was Pac-Man's reasoning for collecting the power-up capsules. Against the '*RandomGhosts*' Pac-Man would collect the power-up capsule and begin hunting Ghosts to maximize his score, whereas against the '*RandomGhost*' agents it appeared as though he would get power-up capsule so that the Ghosts would leave him alone and he could eat more food pellets. This was evidenced in our weights graph as well, as we saw that the weight for eating food grew incredibly large while the weight for eating Ghosts was quite small in comparison.

IX. Conclusion

Our group was able to train a Pac-Man agent that was able to win games by collecting pellets and avoiding Ghosts. Starting with a traditional Q-learning approach, Pac-Man was able to learn how to avoid the Ghost to collect a pellet to complete the game. However, this was with only a small layout and a fairly small state space. As our layouts got bigger, exponentially so became our state space. To solve this, we implemented an Q-value approximation that took features present in the environment and the agent learned how to play based on those features. We found this to be successful and even saw Pac-Man exhibit different behavior based on which Ghost it was trained on.

For future work, we could experiment with more features to make Pac-Man even more successful against the harder Ghost agents. This could include detecting if it's coming close to a dead end where the Ghosts could trap it and end the game. We would also like to model a more accurate representation of Pac-Man, as the original game does not have all four Ghosts function exactly the same way. With more time, it could be interesting to see how Pac-Man performs with a variety of different Ghost agents as opposed to training against the same type of Ghost behavior for every episode.

X. Graphs

The following graphs were all made with the '*mediumClassic*' layout. Three sets of graphs for each '*RandomGhost*' and '*DirectionalGhost*'

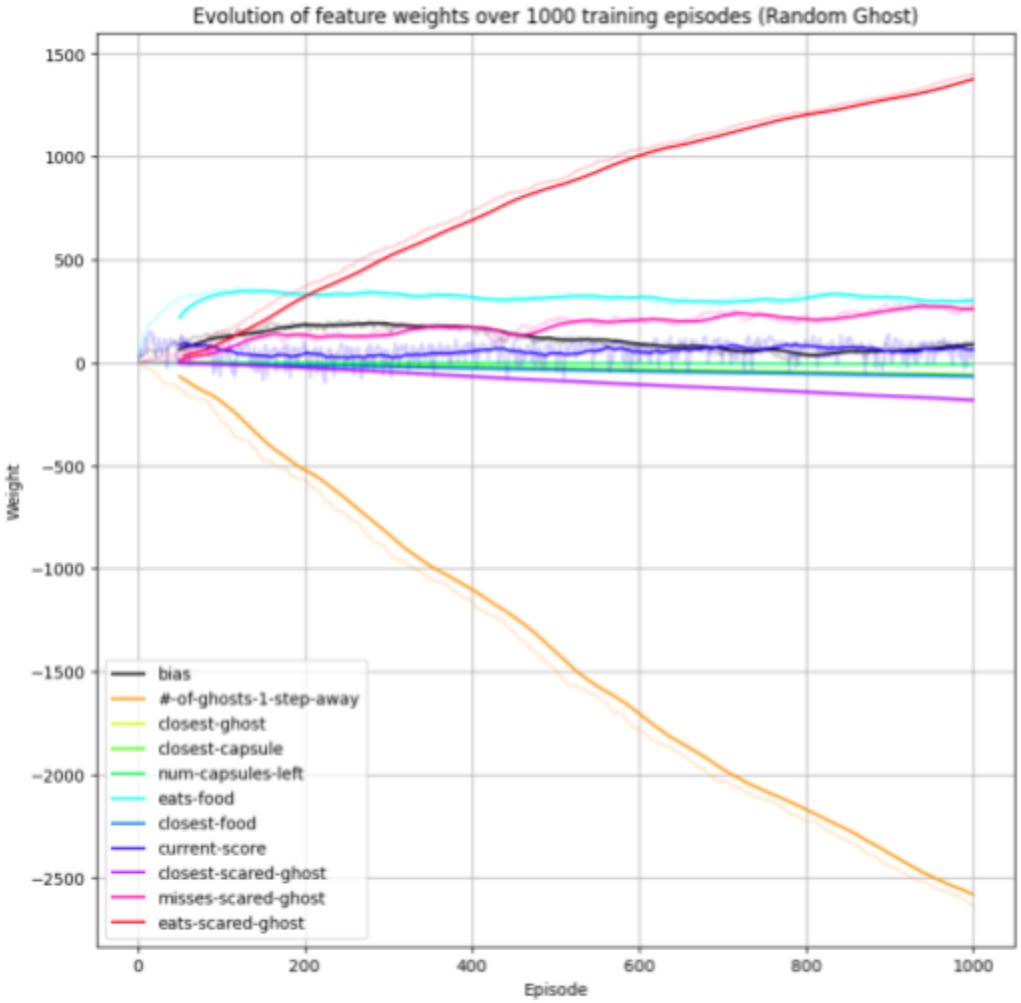

fig 6, feature weights over 1000 training episodes with random ghost behavior

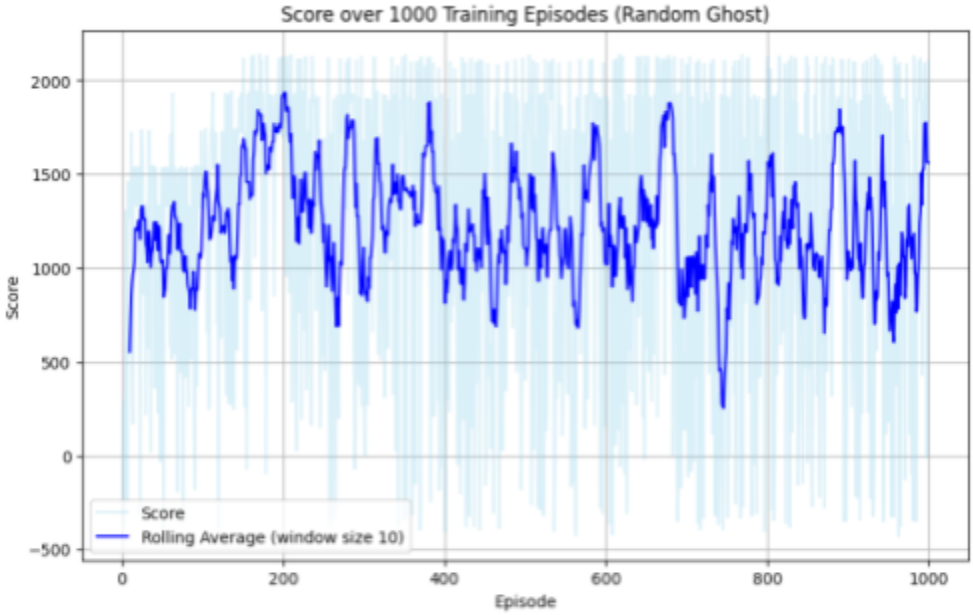

fig 7, average score over 1000 training episodes

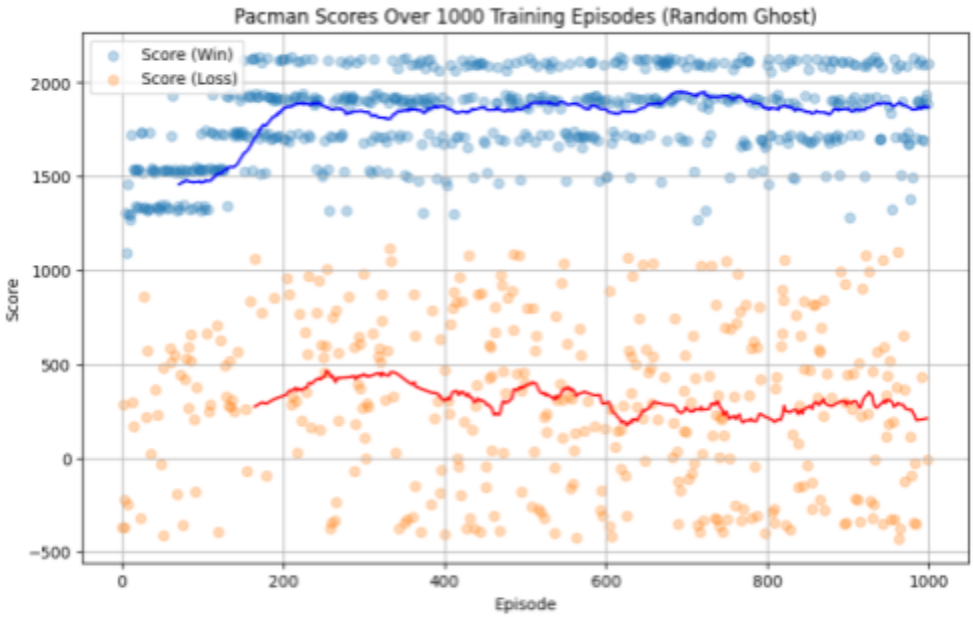

fig 8, average score over 1000 training episodes (with separated win and loss averages)

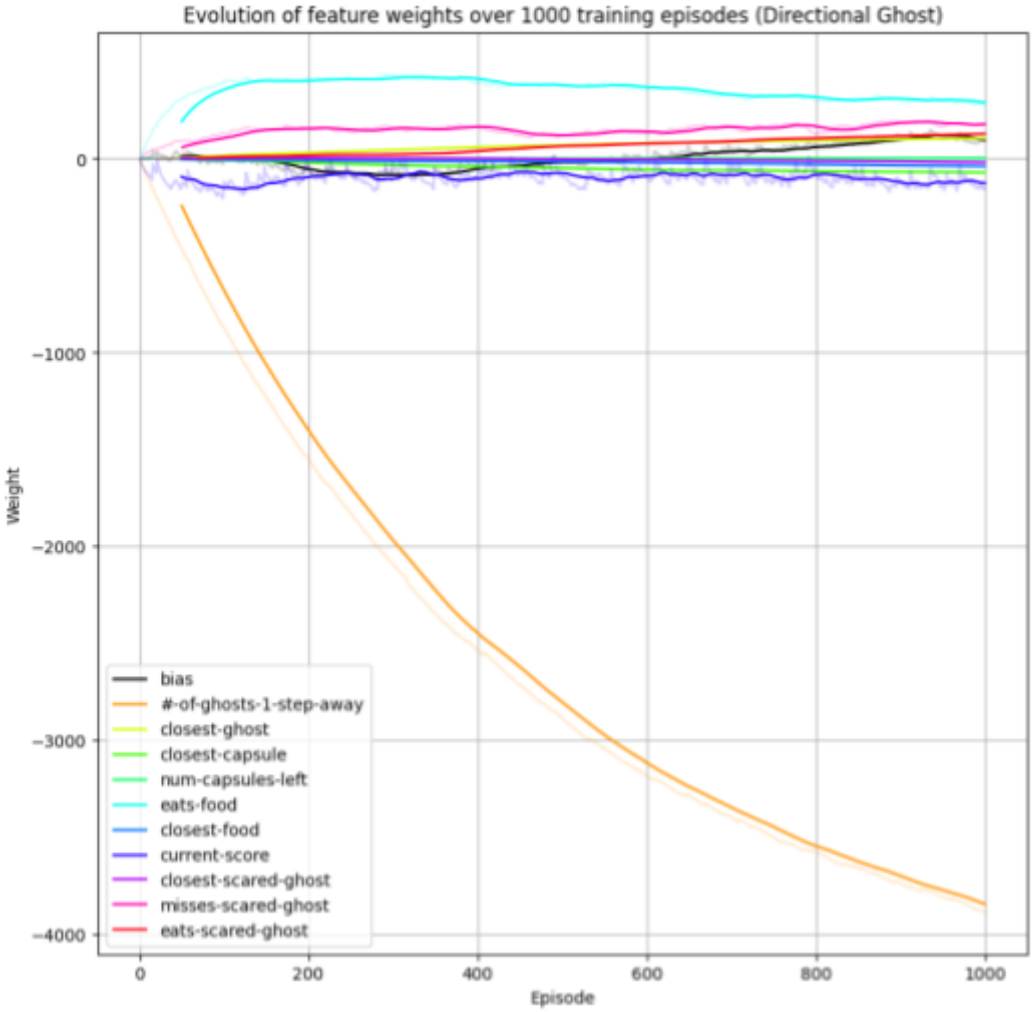

fig 9, feature weights over 1000 training episodes with directional ghost behavior

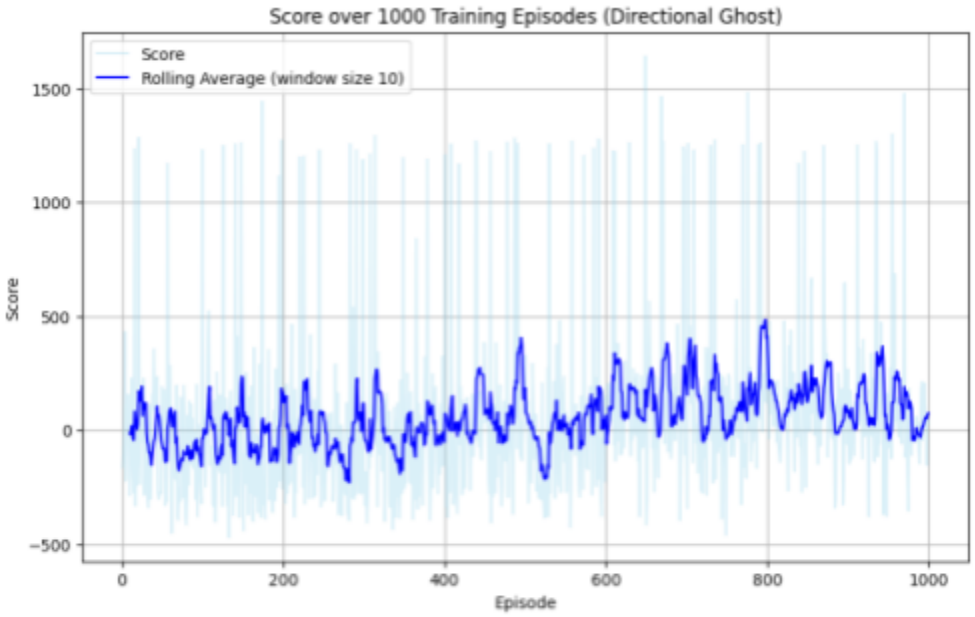

fig 10, average score over 1000 training episodes

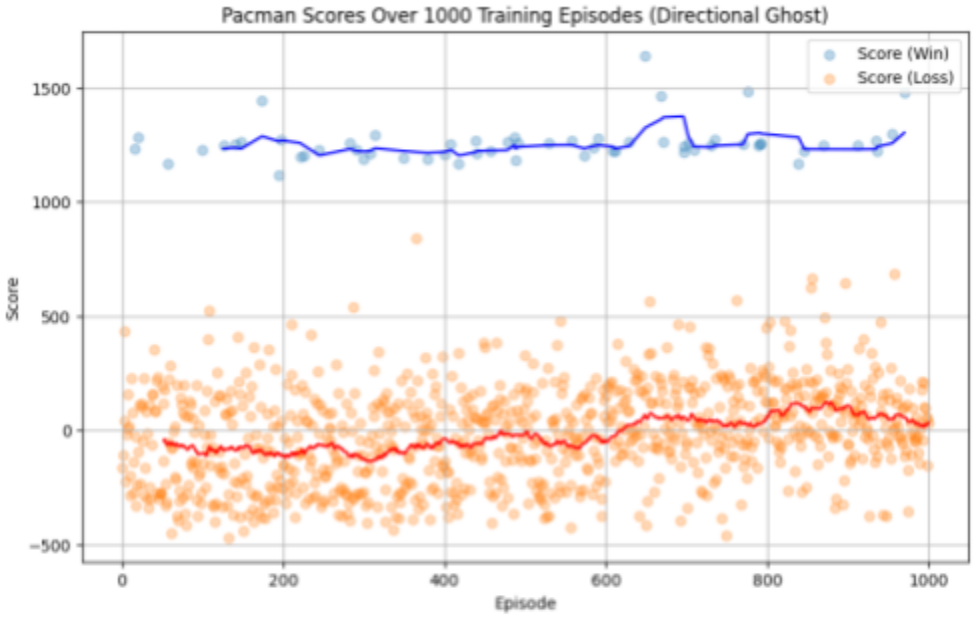

fig 11, average score over 1000 training episodes (with separated win and loss averages)

XI. Related Works

a. **"PAC-MAN Meets Deep Reinforcement Learning" by Çoban et al. (2020):** This paper adopts a deep Q-learning algorithm, a combination of deep learning and reinforcement learning, to tackle the Pac-Man game. The main challenge addressed by this paper is the large state-action space problem inherent in games like Pac-Man, where the environment's state can change dramatically based on the agent's actions and the adversary's (Ghosts) movements. To overcome this, the authors use a deep neural network to estimate the Q-values, which allows generalization across similar states and thus significantly reduces the state-action space. They demonstrated that

their agent could learn to make decisions based on the current game state and showed the potential for deep reinforcement learning approaches in complex game environments.

b. **"Ms. Pac-Man Versus Ghost Team CIG 2016 Competition" by Rohlfshagen et al. (2016):** This research presents a competition involving multiple artificial intelligence agents to play the game of Ms. Pac-Man. The game's complexity is increased in this variation, as the Ghosts in the game use diverse strategies, forcing the Ms. Pac-Man agent to continually adapt. The authors mention several teams employed reinforcement learning, including Q-Learning, for their agents. This study demonstrates that reinforcement learning is capable of adjusting to dynamic and uncertain environments, which are typical characteristics in many games, including Pac-Man.

c. **"Efficient Reinforcement Learning for High Dimensional Linear Quadratic Systems" by Zhang et al. (2020):** While not directly related to the game of Pac-Man, this paper discusses reinforcement learning's efficiency in high dimensional systems. The authors propose a novel reinforcement learning algorithm that achieves better computational and statistical efficiency. This research provides valuable insights that can be applied to our Pac-Man problem, especially considering the high dimensionality due to the complex grid-world environment. We could potentially adopt similar efficient reinforcement learning strategies for our problem, allowing us to handle the Pac-Man game's complexity in a more robust manner

XII. Acknowledgements

We would like to thank Professor James Marden and Mr. Bryce Ferguson for their teaching and mentorship through the quarter and this project. We would also like to thank our classmates for listening to our presentation and inspiring some of the improvements that we have made to our project and models since then.