



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Designing Artificial Intelligence Agents for Pacman Game with Reinforced Learning

By

Ranhao Xu

Supervisor: Mr. Michael Dacey

Project submitted as part of the requirements for the award of

MSc Software Engineering and Artificial Intelligence

August 2024

DECLARATION

I, ...Ranhao Xu..... declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed : Ranhao Xu.....

Date : 28/08/2024.....

FORM OF CONSENT

I Ranhao Xu..... hereby consent that my Project, submitted in candidature for the M.Sc. Applied Computing degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed : Ranhao Xu.....

Date : 28/08/2024.....

Table of Contents

Abstract	4
1. Introduction.....	5
1.1. How to do this project	5
1.2. Why to do this project	7
1.3. Expected outcomes	7
2. Literature Review	8
2.1 Classic Q-learning.....	8
2.2 Approximate Q-learning.....	9
2.3 Q-learning in Pacman.....	10
3. Methodology	12
3.1. Game Framework	12
3.2. Value Iteration.....	12
3.3. Q-Learning.....	15
3.4 Epsilon Greedy.....	17
3.5 Approximate Q-Learning	20
3.6 Deep Q-Learning.....	21
4 Result.....	22
5 Evaluation	29
5.1 Conclusion	29
5.2 Fulfill objectives and aim	31
5.3 Improvement	32
5.4 Overall	33
6 Conclusion	35
7 References.....	36

Abstract

This research project explores the application of Reinforcement Learning (RL) algorithms to the classic Pacman game. The objective is to develop a reinforcement learning agent capable of learning optimal strategies to navigate the game environment, avoid ghosts, and collect as many pellets as possible. By implementing and fine-tuning various reinforcement learning algorithms, including Q-learning and approximate Q-learning, the study aims to assess their performance in dynamic and unpredictable environments like Pacman. The results of this research will provide insights into the effectiveness of reinforcement learning in complex, real-time decision-making tasks, highlighting both its potential and limitations. Additionally, this project will contribute to the understanding of how RL can be adapted to similar problems in other domains, where autonomous agents must learn and act within intricate environments.

1.Introduction

The Pacman game, a classic arcade title, has long served as a benchmark for artificial intelligence research. Its complex environment, requiring agents to balance exploration, exploitation, and risk management, makes it an ideal testbed for reinforcement learning algorithms. This research project explores the application of reinforcement learning to the Pacman game, aiming to develop an agent capable of learning optimal strategies to navigate the maze, avoid ghosts, and maximize its score.

Reinforcement learning is a machine learning paradigm that enables agents to learn optimal policies through interaction with an environment. Unlike supervised learning, which relies on labeled data, reinforcement learning agents learn from experience by receiving rewards or penalties based on their actions. The goal of a reinforcement learning agent is to maximize its cumulative reward over time by selecting actions that lead to desirable outcomes. The aim of this research project is to contribute to the field of reinforcement learning by demonstrating its applicability to challenging game environments and providing insights into the development of intelligent agents capable of adapting to dynamic and uncertain situations.

1.1. How to do this project

In the context of Pacman, the agent's environment is the game board, and its actions are the movements it can make. The agent receives rewards for collecting pellets and penalties for being eaten by a ghost. By trial and error, the agent learns to associate actions with outcomes and develops a policy that guides its behavior.

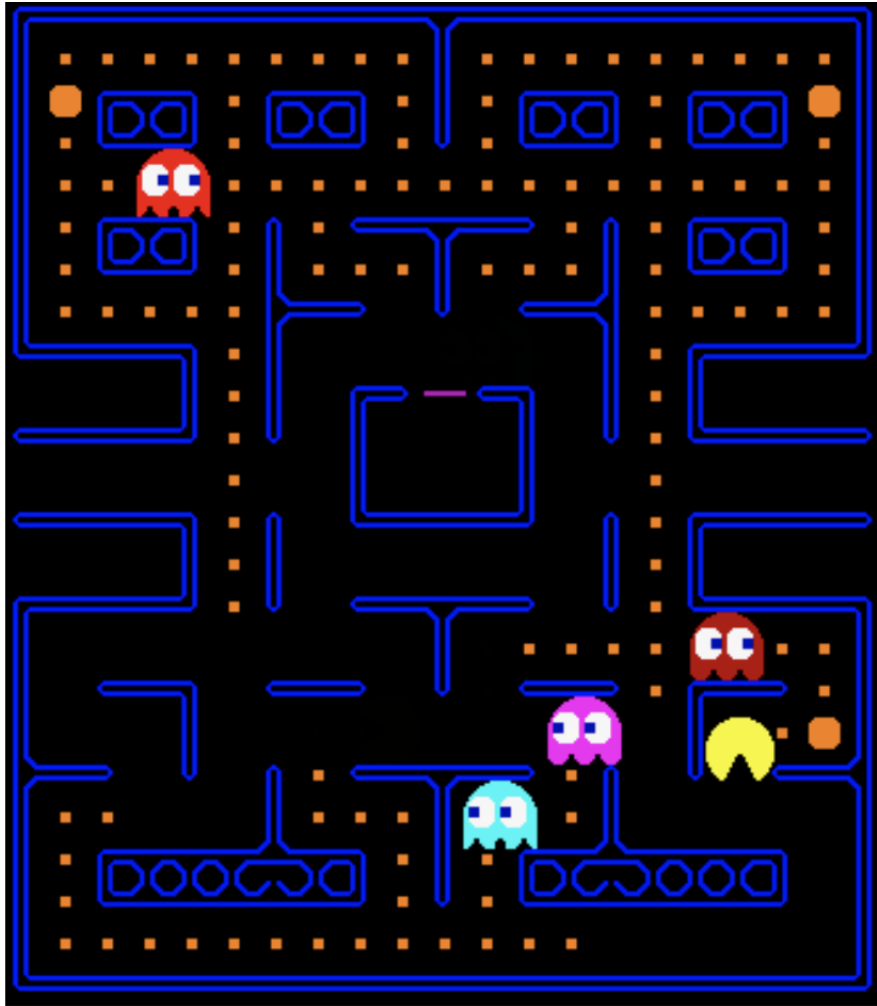


Figure 1. Complex Pacman game snapshot [1]

The reinforcement learning process involves three main components: the agent, the environment, and the reward function. The agent is the decision-maker that selects actions based on its current state and the observed rewards. The environment is the external world that the agent interacts with, providing feedback in the form of rewards and observations. The reward function defines the objective of the agent by specifying the value of different outcomes.

The agent's learning process is iterative. It starts with an initial policy, which may be random or based on prior knowledge. The agent then interacts with the environment, taking actions and receiving rewards. It uses this experience to update its policy, making it more likely to choose actions that lead to higher rewards. This process continues until the agent converges to a near-optimal policy.

1.2. Why to do this project

There are several compelling reasons to implement reinforcement learning in the Pacman game. Pacman is a well-established benchmark in the field of AI and game development. Its complex environment and dynamic gameplay make it an ideal platform for testing and evaluating reinforcement learning algorithms. By applying reinforcement learning to Pacman, researchers can assess the capabilities of different reinforcement learning techniques and compare their performance to traditional AI approaches.

Pacman requires agents to make complex decisions in a dynamic and uncertain environment. By studying how reinforcement learning agents learn to navigate this environment, researchers can gain insights into the mechanisms of complex decision-making and explore the limitations of current reinforcement learning approaches.

The skills and techniques developed through reinforcement learning in Pacman can be applied to a variety of real-world problems. For example, reinforcement learning agents can be used for autonomous navigation [2] and robotics [3]. The ability to learn from experience and adapt to changing environments is crucial for these applications.

1.3. Expected outcomes

The primary goal is to create a reinforcement learning agent that can consistently outperform baseline strategies in the Pacman game. This agent should be able to effectively navigate the maze, avoid ghosts, and collect as many pellets as possible. The agent should demonstrate a deep understanding of the game's dynamics, including the behavior of the ghosts, the optimal paths through the maze, and the trade-offs between risk and reward.

The research will provide valuable insights into the strengths and limitations of different machine learning algorithms when applied to complex game environments. By analyzing the performance of the agent and comparing it to baseline strategies, we can identify which algorithms are best suited for tasks that require exploration, exploitation, and risk management. These insights can inform future research and applications of reinforcement learning in other domains.

The findings of this research can also contribute to the development of more sophisticated

and engaging game AI. The reinforcement learning agent developed in this project could serve as a foundation for creating more challenging and dynamic game opponents. For example, the agent's ability to learn and adapt could be used to create opponents that can adjust their strategies based on the player's behavior, making the game more engaging and challenging.

By exploring the application of reinforcement learning to Pacman, this research may lead to new theoretical developments in the field of reinforcement learning. For example, we may discover novel techniques for handling complex state spaces, continuous action spaces, or dynamic environments. These advancements could have broader implications for the development of RL algorithms and their application to real-world problems.

2. Literature Review

The application of reinforcement learning to the classic Pacman game has been a fruitful area of research, providing insights into the capabilities and limitations of reinforcement learning algorithms in complex, dynamic environments.

2.1 Classic Q-learning

Watkins & Dayan (1992) pioneered the use of Q-learning for reinforcement learning, laying the foundation for subsequent advancements. [4] Q-learning is a model-free reinforcement learning algorithm that learns a Q-function, which estimates the expected future reward for taking a particular action in a given state. By maximizing the Q-function, the agent can learn to select actions that lead to the highest cumulative reward over time. Watkins & Dayan's work demonstrated the effectiveness of Q-learning in various tasks, including the control of a pole-balancing system and the navigation of a maze. Their contributions have had a profound impact on the field of reinforcement learning and continue to be influential today.

This paper presents and proves in detail a convergence theorem for Q-learning based on that outlined in Watkins (1989). This paper shows that Q-learning converges to the optimum action-values with probability 1 so long as all actions are repeatedly sampled in all states and

the action-values are represented discretely.

2.2 Approximate Q-learning

Approximate Q-learning is a technique used when the Q-value function becomes too large to store explicitly. This is often the case in complex environments with high-dimensional state and action spaces. Several methods have been proposed to approximate the Q-value function.

Function approximation is a technique used in machine learning and other fields to estimate a complex function using a simpler function. Tesauro, G. J. (1995) [5] used a neural network to approximate the Q-value function for a backgammon-playing agent. “TD-Gammon is a neural network that is able to teach itself to play backgammon solely by playing against itself and learning from the results, based on the TD (γ) reinforcement learning algorithm (Sutton, 1988)” [6].

In 2015, a paper [7] published in the prestigious journal Nature made a significant breakthrough in the field of reinforcement learning. The authors developed a deep learning algorithm called Deep Q-Networks (DQN) that was able to learn to play a variety of Atari 2600 games at a human-level performance.

This was a major achievement because it demonstrated that deep reinforcement learning could be used to solve complex tasks that had previously been thought to be beyond the capabilities of artificial intelligence. The DQN algorithm was able to learn directly from raw pixel data, without any prior knowledge of the game's rules or dynamics. It was also able to generalize its knowledge to new games, even if they were very different from the ones it had trained on.

The success of the DQN algorithm has led to a surge of interest in deep reinforcement learning, and it is now being applied to a wide variety of tasks, such as robotics, autonomous driving, and natural language processing.

“An important part of RL research is aimed at improving the performance during this learning phase [8]. One way in which this can be achieved is by introducing bias in the decision process of the agent while it is still taking some random decisions. Sutton et al. achieve this by introducing one or more predefined policies, called “options” that the agent can commit to for some time [9]. A novel algorithm is presented that learns options from demonstrations provided by agents familiar with the environment. The intuition is that options learned from experts are likely to produce a good bias for an RL agent learning phase. Other works on learning options from demonstration have recently been published: based on the frequency of sub-trajectories [10], change of abstractions [11], human interaction [12].”

Most of the work in literature deal with Pong, Atari games in general. In 2018, The deep reinforcement learning community has made several independent improvements to the DQN algorithm. [13] A DQN architecture, Rainbow, has been introduced that incorporates several techniques to significantly improve performance. The Rainbow architecture builds upon the original DQN framework and combines Prioritized Experience Replay, Double Q-Learning, Dueling Network Architectures, Noisy Networks, and Multi-Step Learning. By combining these techniques, Rainbow achieves state-of-the-art performance on a variety of Atari 2600 games. It demonstrates the power of deep RL and the importance of carefully engineering DQN architectures to improve performance.

2.3 Q-learning in Pacman

Pacman has long been a popular testbed for artificial intelligence research, and Q-learning has been a particularly effective approach for training agents to navigate its complex environment. In "Reinforcement Learning in Pacman" by Abeynaya Gnanasekaran, Jordi Feliu Faba, and Jing An [14], reinforcement learning approaches are utilized by the authors to

the Pacman game. The goal is to train a Pacman agent to perform optimally by avoiding ghosts, eating food, and maximizing its score.

The authors conducted experiments on various Pacman grid layouts, comparing the performance of Q-learning, approximate Q-learning, and DQN. As a result, the Q-learning performed well on small grids but struggled with larger layouts due to the curse of dimensionality. Approximate Q-learning improved performance by using hand-crafted features, but still faced limitations in representing complex state-action relationships. DQN significantly outperformed both Q-learning and approximate Q-learning, demonstrating its ability to learn effective policies in complex environments.

In their findings, DQN is a powerful approach for reinforcement learning in Pacman, capable of learning optimal or near-optimal policies. The choice of features in approximate Q-learning is crucial for its performance, but designing good features can be challenging. DQN's ability to automatically learn relevant features from raw game data is a key advantage.

The authors also explored different hyperparameters and network architectures to optimize DQN's performance. They also considered the impact of experience replay and target networks, which are essential components of DQN. Overall, this paper provides a valuable contribution to the field of reinforcement learning by demonstrating the effectiveness of DQN in solving challenging problems like Pacman.

3. Methodology

3.1. Game Framework

For the experiments, an established python version of Pacman is used. The Pac-Man projects were developed for CS 188 at UC Berkeley, primarily by John DeNero and Dan Klein. [15]. They apply an array of AI techniques to playing Pac-Man. The project allows you to visualize the results of the techniques you implement. The Pacman project consists of several key components. It keeps track of the state of the game, as well as the states of Pacman and ghost agents, including configuration, speed, scared, etc. The agent can be controlled by keyboard. The output from the reinforcement learning algorithms can replace the keyboard entry for controlling the agent.

There are several different layouts of the game board for different development requirements.

3.2. Value Iteration

Value Iteration is a dynamic programming algorithm used to compute the optimal value function in a Markov Decision Process (MDP). It's a popular method for solving MDPs because it's relatively simple to implement and can be applied to a wide range of problems. The value iteration agent is an offline planner, not a reinforcement learning agent, and so the relevant training option is the number of iterations of value iteration it should run in its initial planning phase. Value iteration computes k -step estimates of the optimal values V_k . Below is the value iteration state update equation:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

where:

$V_{k+1}(s)$ is the estimated value of state s at iteration $k + 1$.

$V_k(s')$ is the value of the next state s' at iteration k .

$\max_a$ denotes that we are selecting the action a that maximizes the expected value.

$T(s, a, s')$ is the probability of transitioning to state s' from state s after taking action a .

$R(s, a, s')$ is the immediate reward received after transitioning from state s to state s' via action a .

γ is the discount factor, which determines the importance of future rewards (where $0 \leq \gamma \leq 1$).

The value iteration algorithm updates the value of each state based on the expected rewards of the possible future states. It uses the Bellman equation to find the value of the best possible action (policy) that can be taken from each state. This process is repeated until the value function converges, meaning the changes between iterations are smaller than a predefined threshold. The value iteration agent takes an MDP on construction and runs value iteration for the specified number of iterations before the constructor returns.

The value iteration algorithm can be tested in a grid world. The grid world is designed as follows:

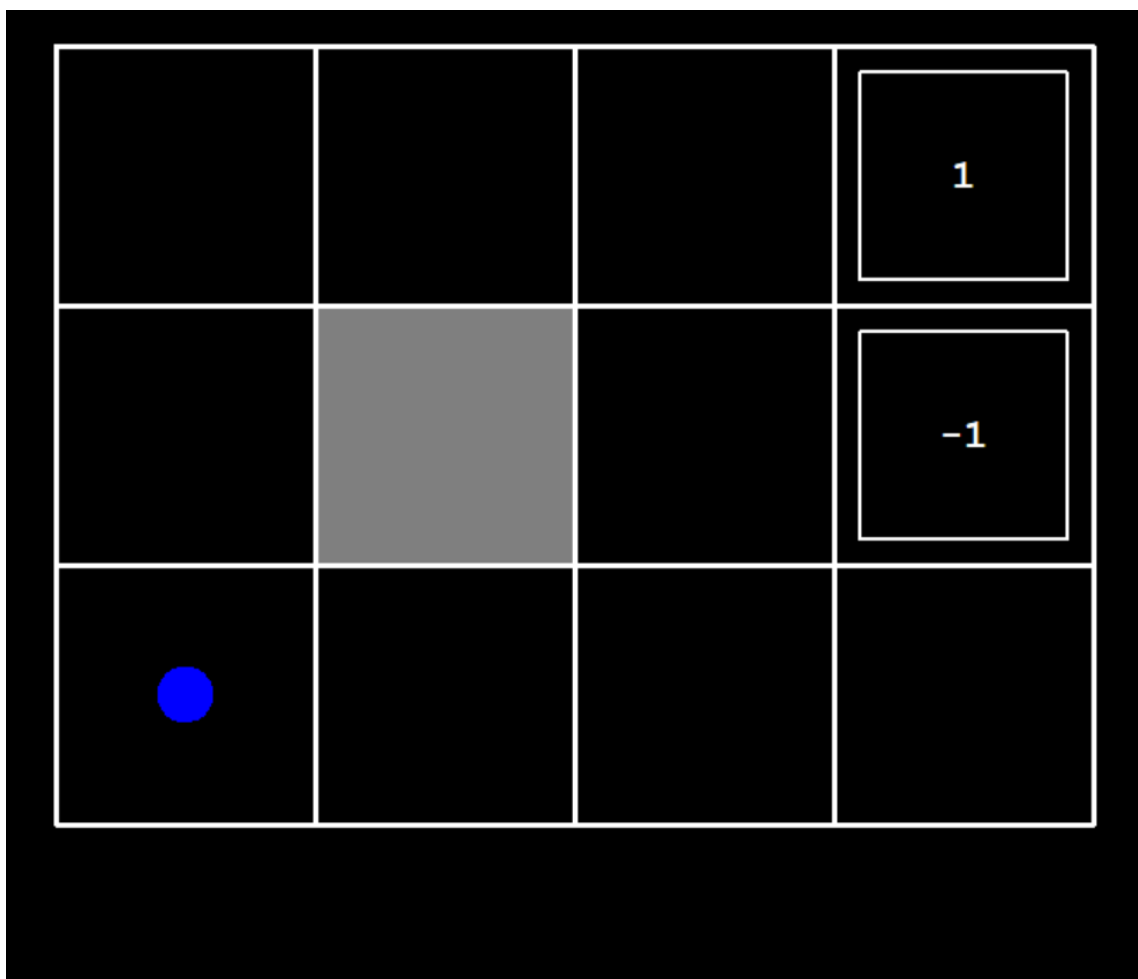


Figure 2. Grid world

where the blue dot represents the agent, grey grid represents the wall, and grids with 1 and -1 represent the exits with the respective reward. The goal for the value iteration is to generate a complete policy with expected reward through iterative training. The result is shown below, where the arrow in each grid represents the best policy:

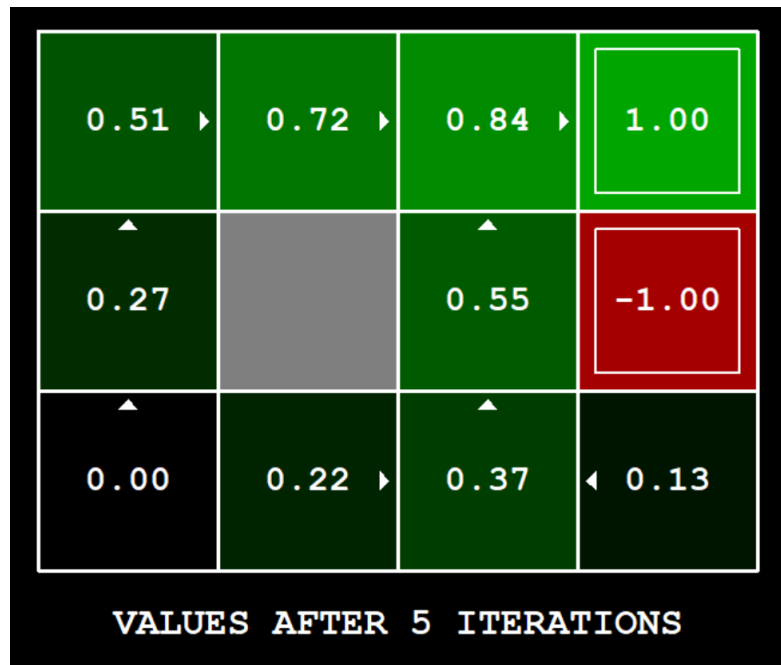


Figure 3. Grid world after 5 iterations



Figure 4. Grid world after 100 iterations

3.3. Q-Learning

The value iteration agent does not actually learn from experience. Rather, it iteratively uses the MDP model to compute a complete policy before starting to interact with the environment. And when it does interact with the environment, it simply follows the precomputed policy. In comparison, the Q-learning agent does very little on construction, but instead learns by trial and error from interactions with the environment through its movements.

Q-learning is a popular model-free reinforcement learning algorithm used to learn the optimal action-selection policy for an agent interacting with an environment. It is designed to find the best action to take given the current state of the environment, aiming to maximize the total cumulative reward over time.

In Q-learning, the core idea is to learn a function, $Q(s, a)$, which estimates the expected utility (or value) of taking action a in state s , and then following the optimal policy thereafter. The Q-value represents the quality of a particular action in a given state. State (s) is the current situation or configuration in which the agent finds itself. Action (a) is a possible move or decision the agent can make in state s . Reward (r) is the immediate feedback received after taking an action a in state s . Discount Factor (γ) is a value between 0 and 1 that represents the importance of future rewards compared to immediate rewards. A discount factor closes to 1 means future rewards are considered almost as important as immediate ones. Learning Rate (α) is a value between 0 and 1 that controls how much the new information overrides the old information. A high learning rate means the agent is more eager to learn from new experiences.

The Q-learning algorithm works by iteratively updating the Q-values using the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Current Q-value $Q(s, a)$ is the current estimate of the value of taking action a in state s . Immediate Reward $R(s, a)$ is the reward received after taking action a in state s . Max Future Q-value $\max_{a'} Q(s', a')$ is the highest Q-value for the next state s' (after taking action

a) over all possible actions a' . This term represents the agent's best estimate of future rewards starting from the next state. The term $R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)$ is the Temporal Difference (TD) error, which measures the difference between the current Q-value and the estimated optimal future Q-value. The agent uses this difference to adjust its estimate. The Learning Rate α determines how quickly the Q-value updates, balancing new information with old estimates.

The process of Q-Learning proceeds as follow. First, the algorithm initializes an empty Q table that stores arbitrary Q-value for each state-action pair. Then the agent interacts with the environment by taking actions, observing the resulting state and reward. After each action, update the Q-value for the state-action pair using the update rule. The process of interacting and updating repeats over many episodes, with the agent continually improving its estimates of the Q-values. Once the Q-values converge, or stop changing significantly, the optimal policy can be derived by choosing the action with the highest Q-value in each state.

The same grid used in value iteration test can also be utilized as a Q-Learning test with a small twist. Instead of generating a single expected reward in each grid, each state and action pair with their respective Q-value is stored and a complete Q-table is generated as the result of training. The result is shown below:

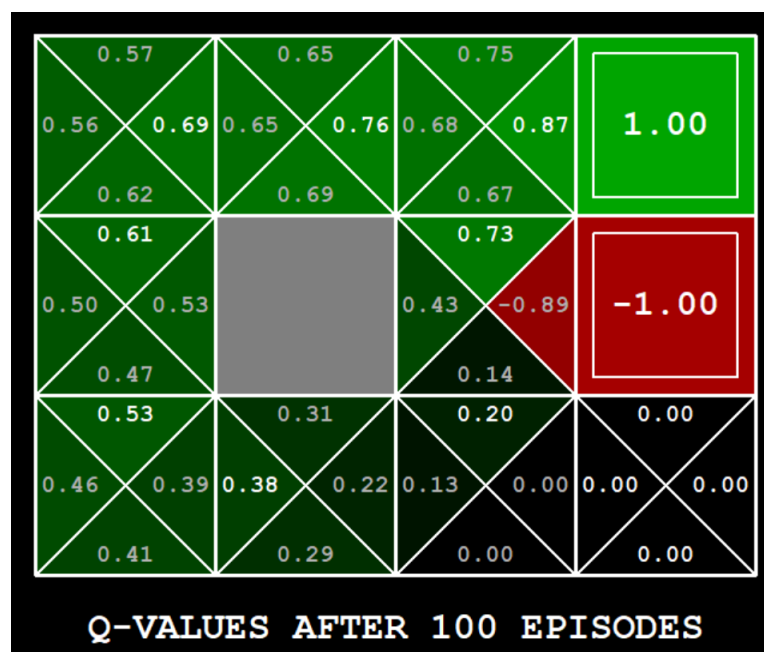


Figure 5. Grid world after 100 episodes

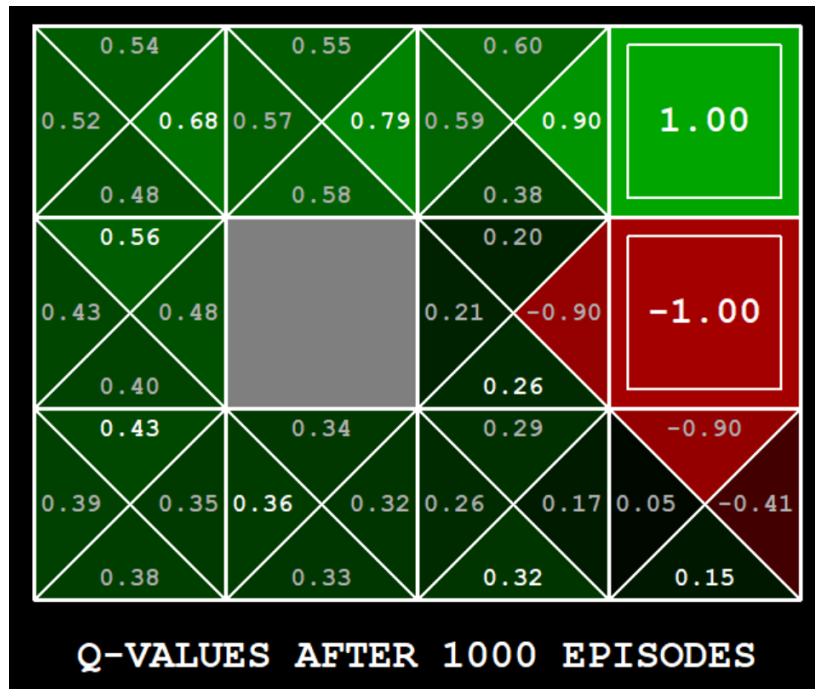


Figure 5. Grid world after 1000 episodes

3.4 Epsilon Greedy

Epsilon-greedy is a popular exploration-exploitation strategy used in reinforcement learning. It provides a balance between exploring new actions and exploiting known good actions. The challenge in reinforcement learning is to find the right balance between exploration and exploitation. If an agent only exploits, it may miss out on discovering better strategies. Conversely, if it only explores, it may not make the most of its current knowledge. By carefully selecting the epsilon parameter, the agent can effectively tradeoff between short-term rewards and long-term exploration.

The epsilon-greedy strategy introduces randomness into the action-selection process to maintain a balance between exploration and exploitation. The epsilon parameter controls the balance between exploration and exploitation. A higher epsilon value leads to more exploration, while a lower epsilon value leads to more exploitation.

With probability ϵ (epsilon), randomly select an action from the available action space. This encourages exploration and helps the agent discover new, potentially rewarding actions. By randomly choosing actions, the agent can explore different parts of the state space and avoid getting stuck in local optima. With probability $1 - \epsilon$, select the action that has the

highest estimated Q-value. This exploits the agent's current knowledge and focuses on maximizing reward. By choosing the best action, the agent can leverage its previous experiences and make more informed decisions.

The difference between high epsilon value and low epsilon value is demonstrated as below:

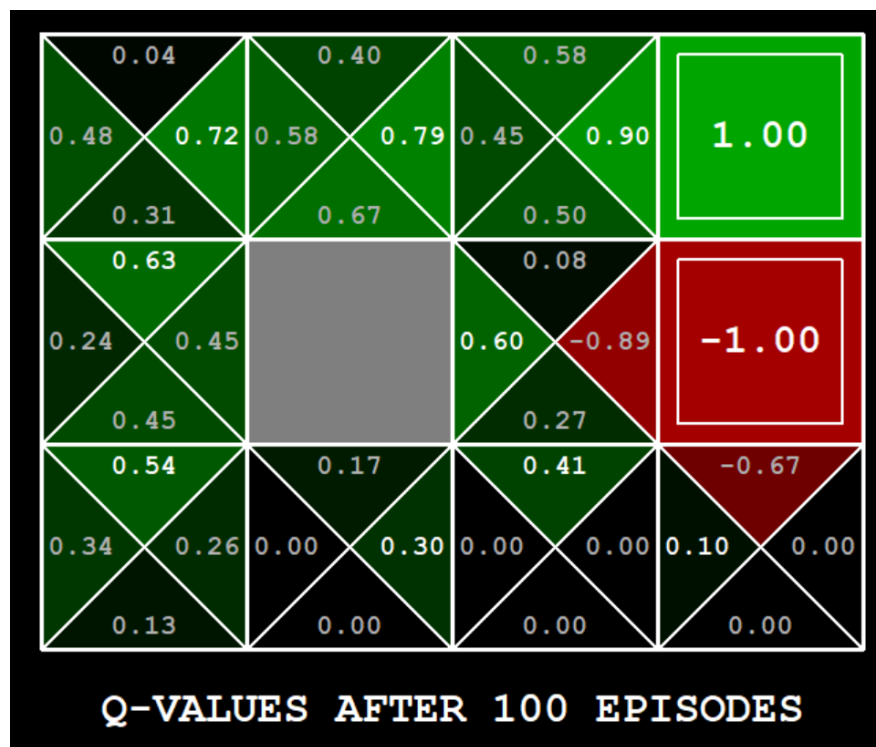


Figure 7. Grid world after 100 episodes

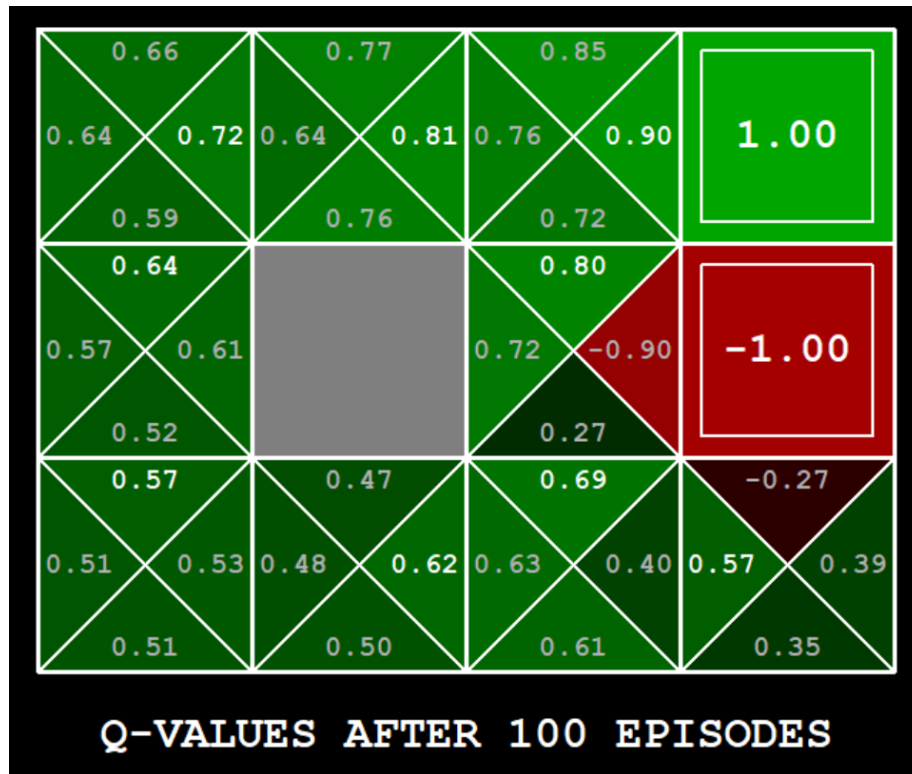


Figure 8. Grid world after 1000 episodes

By visually comparing these two Q-tables, a conclusion can be made that a Q-learning agent explores more possibilities with higher epsilon value than with lower epsilon value. After 100 episodes of training with $\epsilon = 0.1$, some of the Q-values in the table still remain as the initial value, where the Q-table acquired with $\epsilon = 0.9$, all the Q-values are filled with expected reward for that specific state and action pair.

An optional step can be taken to strike a balance between exploration and exploitation over time, decay the epsilon value gradually. This means that the agent explores more at the beginning of learning and gradually shifts towards exploitation as it gains more experience.

3.5 Approximate Q-Learning

In traditional Q-learning, the agent maintains a Q-table that stores Q-values for every possible state-action pair. This approach works well for environments with small, discrete state and action spaces. However, in more complex environments with large or continuous spaces, the Q-table becomes too large to store or update efficiently. Approximate Q-learning addresses this issue by using function approximation to estimate the Q-values.

Consider a linear function approximation:

$$Q(s, a) = \sum_{i=1}^n f_i(s, a)w_i$$

where each weight w_i associated with a particular feature $f_i(s, a)$. The weight vector is implemented as a dictionary mapping features to weight values.

The weight vectors are updated similarly to the Q-values:

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s, a)$$
$$\text{difference} = \left(r + \gamma \max_{a'} Q(s', a') \right) - Q(s, a)$$

where r is the experienced reward. The difference is also known as the temporal difference error. The learning rate α is used to control how much the parameters are adjusted during each update.

The Approximate Q-Learning can effectively handle environments with vast state spaces, where traditional Q-learning would be impractical due to the memory requirements of storing a Q-table for every possible state-action pair. By generalizing from past experiences, the function approximator can estimate Q-values for new states that it has not encountered before, allowing the agent to adapt to changing environments and make informed decisions in novel situations.

Approximate Q-learning is also scalable to large-scale problems. By using function approximators, the agent can learn from a large amount of data and generalize to new situations. This makes approximate Q-learning suitable for real-world applications, such as robotics, autonomous driving, and game playing, where the environment is complex and constantly changing.

3.6 Deep Q-Learning

Deep Q-Learning is an advancement from Approximate Q-learning algorithm. It is a cutting-edge reinforcement learning algorithm that combines the power of deep learning with the traditional Q-learning framework. By leveraging deep neural networks, Deep Q-Learning is capable of handling high-dimensional state spaces, making it particularly well-suited for complex tasks such as playing video games, controlling robots, and navigating challenging environments.

One of the key advantages of Deep Q-Learning is its ability to generalize to unseen states. Unlike traditional Q-learning, which requires a separate Q-value for each possible state-action pair, Deep Q-Learning can estimate Q-values for new states by leveraging the patterns and relationships learned from previous experiences. This generalization capability is essential for tasks with large or infinite state spaces, where it would be impractical to store a Q-value for every possible combination of state and action.

Furthermore, Deep Q-Learning incorporates several techniques to improve its performance and stability. The use of experience replay helps to reduce correlation between consecutive samples, leading to more efficient learning. The target network, a separate neural network used to estimate the target Q-values, helps to reduce overestimation bias and improve convergence. These techniques, combined with the power of deep neural networks, make Deep Q-Learning a highly effective and versatile reinforcement learning algorithm.

There are also challenges of Deep Q-Learning. Training deep neural networks can be computationally expensive, requiring significant processing power and large datasets. Hyperparameter tuning, like learning rates and network architectures, is another complex task for optimal performance.

4 Result

In this section, we present and analyze the outcomes of applying reinforcement learning, specifically Q-learning and its approximate variants, to the game of Pacman. The results highlight the agent's learning process, performance improvements over time, and the effectiveness of different configurations and algorithms. The performance of the RL agent will be evaluated through extensive simulations. Metrics such as the average game score, and the agent's survival rate, learning curve, and ability to handle large state space will be used to assess the effectiveness of the RL approach.

There are several different layouts used for the training and testing section: small grid, medium grid, small classic, medium classic, original classic.

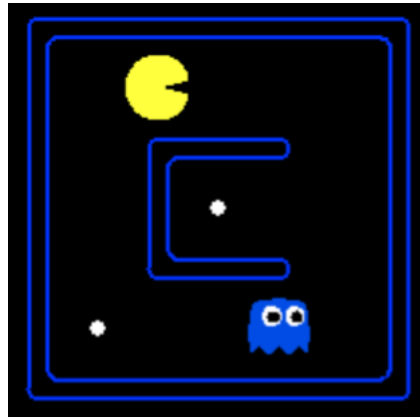


Figure 9. Small grid layout

For the small grid layout, Q-learning agent and Approximate Q-learning agent are both tested and evaluated. The training process is set to 2000 episodes, and 50 games is simulated after training. Below is the result for the simulations:

Agent	Layout	Training Episodes	Test Runs	Average Score	Win Rate
Q-Learning	SmallGrid	2000	50	398.14	90%
Approximate Q-Learning	SmallGrid	2000	50	500.04	100%

Figure 10. Simulation result on small grid layout

The provided results demonstrate a clear improvement in performance when using Approximate Q-Learning compared to traditional Q-Learning on a small grid Pacman environment. Approximate Q-Learning significantly outperforms Q-Learning in terms of the average score. This indicates that the function approximator is able to generalize better and learn more effective policies. The feature extractor is a very simple approximator that adjusts

the feature based on the result of next Pacman movement: whether the movement will result an increase in the game score, and whether the movement will result in a termination of the game.

Approximate Q-Learning achieves a 100% win rate, suggesting that it is able to consistently navigate the environment and avoid ghosts. In contrast, Q-Learning has a win rate of 90%, indicating that it may still struggle in certain scenarios or insufficient learning.

The learning curves for Q-learning and Approximate Q-learning over 2000 training episodes are shown below:

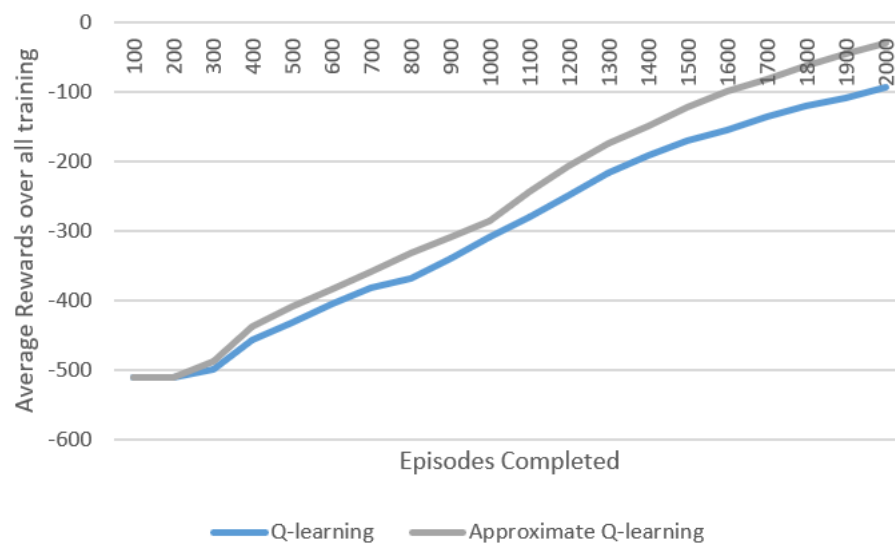


Figure 11. Learning curve of episodes completed vs. average rewards over all training

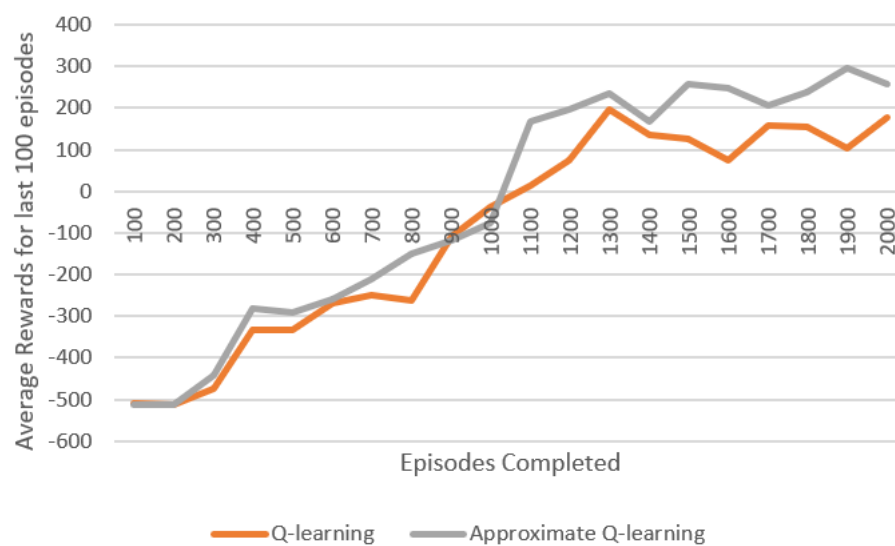


Figure 12. Learning curve of episodes completed vs. average rewards over last 100 episodes

The provided figures show two learning curves representing the average rewards over all training and over the last 100 training episodes for Q-learning and Approximate Q-learning.

Both Q-learning and Approximate Q-learning start with negative average rewards, indicating that the agents are initially making suboptimal decisions. This is expected, as both algorithms begin with random or arbitrary Q-values and need to explore the environment to learn effective strategies. As training progresses, both algorithms gradually improve their performance, with the average rewards increasing over time. This indicates that the agents are learning from their experiences and making more informed decisions.

Approximate Q-learning consistently outperforms Q-learning in terms of average rewards. This suggests that the function approximator used in Approximate Q-learning is able to generalize better and learn more effective policies. The ability to generalize allows Approximate Q-learning to handle larger state spaces and learn from a wider range of experiences, leading to improved performance.

Approximate Q-learning appears to converge more quickly than Q-learning. This could be due to the function approximator's ability to learn from a larger number of experiences and generalize to new situations. By leveraging the power of function approximation, Approximate Q-learning can identify patterns and relationships in the data more efficiently, leading to faster convergence to optimal policies.

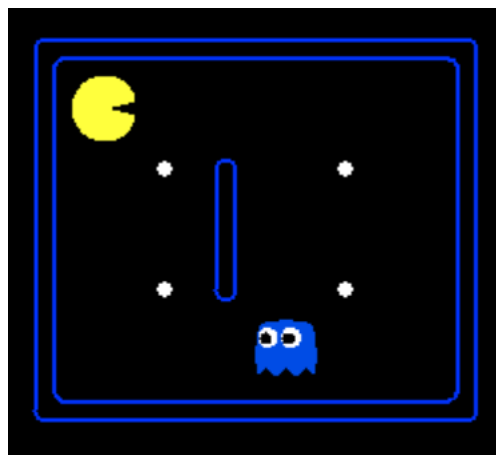


Figure 13. Medium grid layout

The next experiment runs the Q-learning agent and Approximate Q-learning agent with different feature extractors against the medium grid layout. The identity extractor assigns a single feature to every state and action pair

The following table shows the result of running Q-learning agent and Approximate Q-learning agent with different feature extractors against the medium grid layout:

Agent	Layout	Training Episodes	Test Runs	Average Score	Win Rate
Q-Learning	MediumGrid	200	50	-507.92	0%
Approximate Q-Learning	MediumGrid	200	50	527.52	100%

Figure 14. Simulation result on medium grid layout

From the table above, it is indicated that both the Q-Larning agent fails to learn from the training effectively on the medium grid layout. The failure of Q-learning agent could be attributed to several factors. Medium grid layouts are generally more complex than small grids, featuring larger mazes, more intricate pathways, and a greater number of potential obstacles. This increased complexity can lead to a larger state space, making it more difficult for the agent to explore and learn effective strategies.

As the state space grows larger, the Q-table can become prohibitively large, leading to computational challenges and potential convergence issues. In the context of Pacman, the state space is defined by the agent's position, the positions of the ghosts, the distances to pellets, and other relevant factors. As the maze size increases, the number of possible states grows exponentially, making it difficult for the agent to store and update Q-values for all of them.

The Pacman environment is non-stationary, meaning that the dynamics of the environment can change over time. This is due to the behavior of the ghosts, which can move randomly or pursue the agent. The non-stationary nature of the environment can make it difficult for the agent to learn a fixed policy that is optimal in all situations.

The Pacman game often involves intermittent rewards, meaning that the agent may not receive a reward for every action. This can make it challenging for the agent to learn the long-term consequences of its actions and develop effective strategies.

With an appropriate custom feature extractor, the Approximate Q-learning agent can learn to win with ease with only 200 training episodes. This extractor is designed for a simple reflex Pacman. First, it extracts the grid of food and wall locations and get the ghost locations. It then computes the location of Pacman after he takes the action and counts the number of ghosts 1-step away. The feature depends on several parameters: whether food will be eaten, how far away the next food is, whether a ghost collision is imminent, and whether a

ghost is one step away.

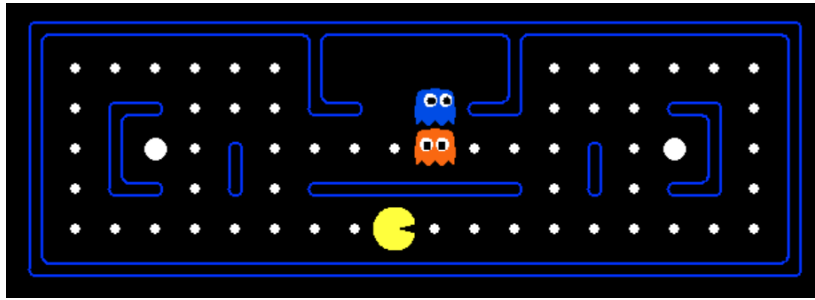


Figure 14. Small classic layout

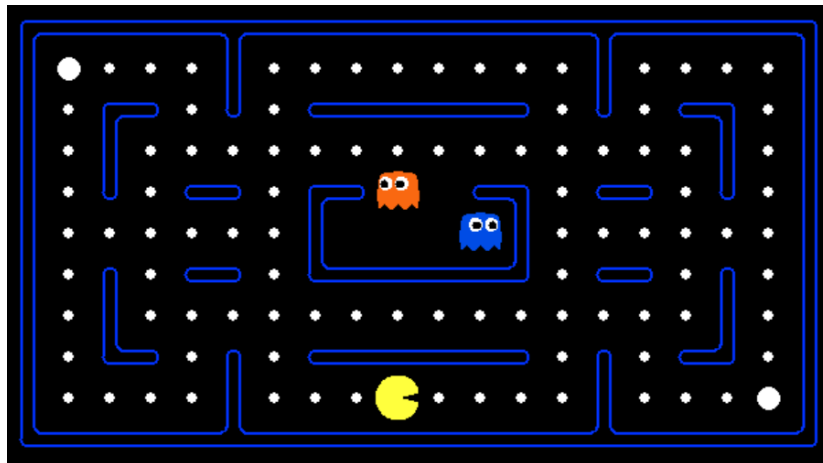


Figure 15. Medium classic layout

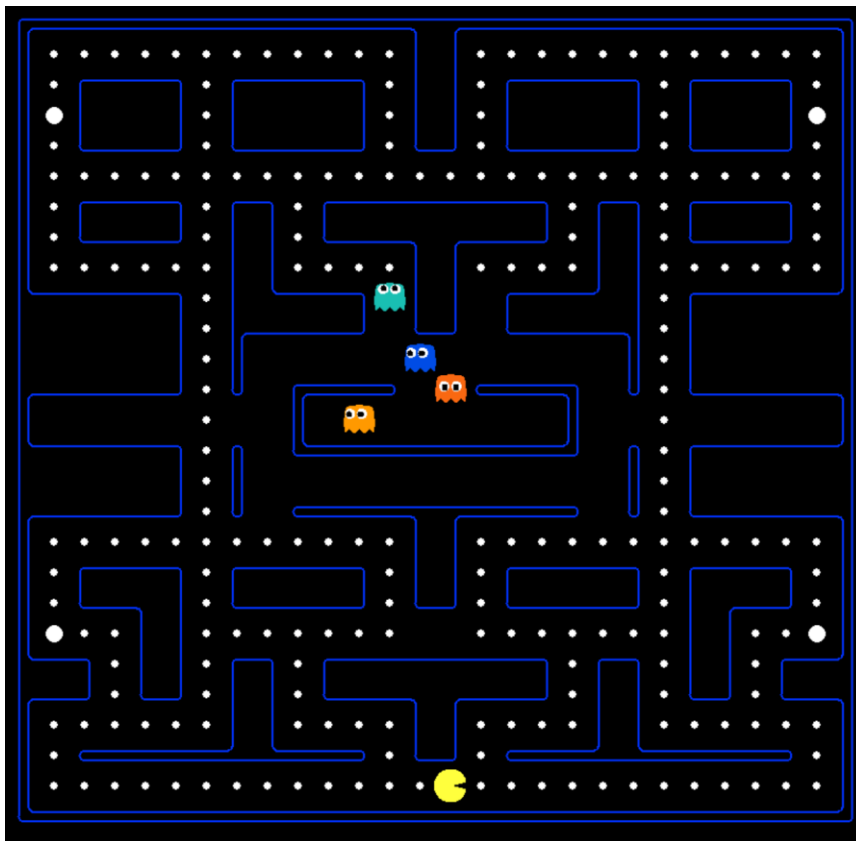


Figure 16. Original classic layout

The following table is the result of running Approximate Q-learning agent against small classic, medium classic, and original classic layouts of the Pacman game.

Agent	Layout	Training Episodes	Test Runs	Average Score	Win Rate
Approximate Q-Learning	SmallClassic	200	50	883.34	92%
Approximate Q-Learning	MediumClassic	200	50	1081.1	80%
Approximate Q-Learning	OriginalClassic	200	50	2135.5	78%

Figure 17. Simulation result on small, medium, and original classic layouts

The provided results demonstrate a clear deterioration in performance when using Approximate Q-Learning on different Pacman layouts. While the win rate decreases slightly as the layout becomes more complex, it remains relatively high, demonstrating the agent's ability to navigate and avoid ghosts effectively in a variety of environments. The decrease in win rate can be attributed to the increased difficulty of the larger layouts, which require more sophisticated strategies and a greater understanding of the game's dynamics. However, the fact that the agent is still able to achieve high win rates on these layouts highlights the effectiveness of Approximate Q-Learning in handling complex reinforcement learning problems.

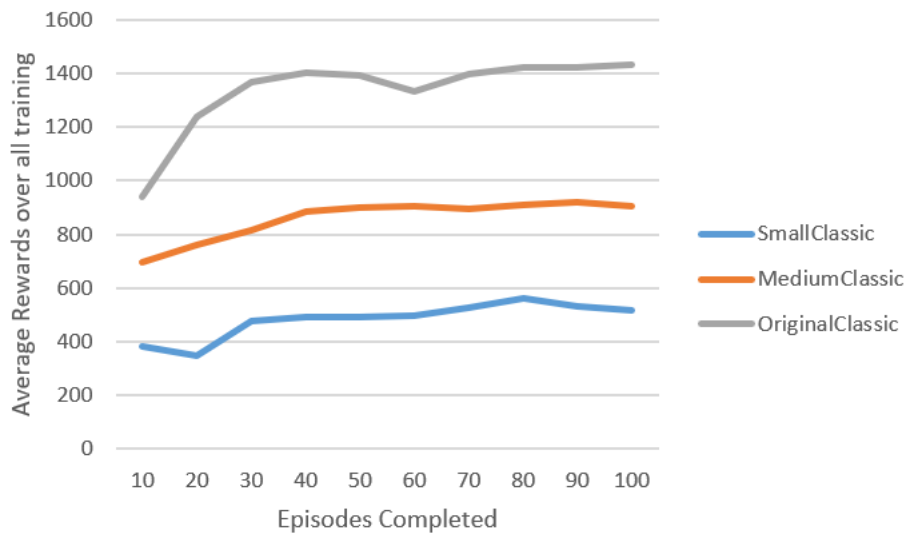


Figure 18. Learning curve of episodes completed vs. average rewards over all training

The provided image shows three learning curves representing the average rewards over all training episodes for Approximate Q-Learning on three different Pacman layouts: small classic, medium classic, and original classic.

All three layouts start with negative average rewards, indicating that the agent is initially making suboptimal decisions. As training progresses, all three layouts demonstrate a gradual

increase in average rewards, suggesting that the agent is learning from its experiences and making more informed decisions. The agent is able to adapt its behavior based on the feedback it receives from the environment, refining its strategies and improving its performance over time. This learning process is essential for mastering the Pacman game and achieving high scores.

The original classic layout consistently achieves the highest average rewards, followed by medium classic and small classic. This indicates that the agent finds the original classic layout to be the most challenging, requiring more sophisticated strategies to navigate and collect pellets effectively. The original classic layout features a larger maze with more intricate pathways, more numerous ghosts, and a denser distribution of pellets. This increased complexity presents a greater challenge for the agent, requiring it to develop more advanced strategies to avoid ghosts, find efficient paths, and maximize its rewards. In contrast, the small classic layout is relatively simple, with a smaller maze and fewer obstacles. This makes it easier for the agent to learn effective strategies and achieve high scores. The medium classic layout falls somewhere in between, offering a moderate level of challenge that requires the agent to balance exploration and exploitation more carefully.

All three layouts appear to have converged to a relatively stable average reward level. This suggests that the agent has learned effective policies for each layout and is no longer making significant improvements. However, it is possible that the agent could continue to make incremental improvements with additional training, especially if the environment is non-stationary or if there are hidden rewards or obstacles that the agent has not yet discovered.

5 Evaluation

5.1 Conclusion

The provided data demonstrates that Approximate Q-Learning is a promising approach for solving Pacman and other complex reinforcement learning problems. There are some conclusions that can be drawn from the result analysis.

First is the effectiveness of Approximate Q-Learning over traditional Q-learning. Approximate Q-Learning consistently outperforms traditional Q-Learning in terms of average score and win rate, even on the most challenging layouts. This indicates that the function approximator used in Approximate Q-Learning is able to generalize better and learn more effective policies, allowing the agent to navigate complex environments and make informed decisions.

The ability of Approximate Q-Learning to handle high-dimensional state spaces and learn from large amounts of data is particularly impressive. This makes it well-suited for real-world applications where the environment is complex and constantly changing.

Second is the impact of the layout complexity. The complexity of the layout has a significant impact on the agent's performance. Larger and more complex layouts require more sophisticated strategies and a deeper understanding of the game's dynamics. Approximate Q-Learning is able to adapt to different layout complexities, demonstrating its versatility and generalizability. However, the agent's performance may still be limited by the complexity of the environment and the available training data.

The third and most important conclusion is that the feature extraction function is crucial to the success of Approximate Q-Learning. Feature extraction plays a crucial role in approximate Q-learning, as it determines the information that the function approximator can use to estimate Q-values. By selecting informative features, the agent can learn more effective policies and generalize better to unseen states.

Well-crafted features can help the agent generalize to unseen states and situations. By capturing relevant information about the environment, the function approximator can learn patterns and relationships that are applicable to a wider range of scenarios. Feature extraction

can reduce the dimensionality of the state space, making it easier for the function approximator to learn and generalize. This can also help to alleviate the curse of dimensionality, which can be a significant challenge in large state spaces. Informative features can help the agent learn more efficiently by providing relevant cues about the environment. By focusing on the most important information, the agent can avoid wasting time on irrelevant details and converge to optimal policies more quickly. Well-chosen features can make it easier to understand and interpret the agent's behavior. By examining the features that the agent is using, we can gain insights into its decision-making process and identify potential areas for improvement.

There are also some challenges in feature extraction. Hand-crafting features can be a time-consuming and labor-intensive process, especially for complex environments. It requires domain expertise to identify relevant features and understand how they interact with each other. Additionally, manual engineering can be prone to human error and may not capture all relevant information.

Determining which features are most relevant to the task can be challenging, especially when the state space is high-dimensional and the relationships between features are complex. Feature selection algorithms can help to identify the most informative features, but they may not always be able to capture all relevant information. Features may interact with each other in complex ways, making it difficult to represent their relationships accurately. For example, the combination of two features may be more informative than either feature alone. This can make it challenging to design effective feature extraction techniques that capture these interactions.

Real-world environments, also in the Pacman project, are often noisy and uncertain, which can make it difficult to extract meaningful features from raw data. Noise can introduce errors into feature extraction, leading to suboptimal performance. Additionally, uncertainty about the environment can make it difficult to determine which features are truly relevant and which are irrelevant.

By carefully selecting and engineering features, the performance of Approximate Q-Learning agent can be significantly improved and enable it to learn more effective policies in

complex environments.

Overall, the results suggest that Approximate Q-Learning is a powerful and effective reinforcement learning algorithm that can be applied to a wide range of challenging tasks. By carefully selecting features, tuning hyperparameters, and considering the specific characteristics of the environment, it is possible to achieve impressive results with Approximate Q-Learning.

5.2 Fulfill objectives and aim

There are three objectives at the beginning of the project. First, successfully completing the project by implementing a reinforcement learning agent that is able to play the Pacman game with some basic layouts. At the end of the project, a functional Approximate Q-Learning agent is implemented and tested against the Pacman game with original classic layout. With an approximate 78% overall win rate after a 200 episodes training, I would say this objective is successfully accomplished.

The second objective is to find reasons to do this project. The Pacman project provides a practical and engaging platform to apply reinforcement learning concepts and algorithms to a real-world problem. By implementing and experimenting with different techniques, I gained a deeper understanding of how reinforcement learning works and its limitations. It offers a solid foundation in core reinforcement learning concepts.

The third objective is to learn something and provide valuable insights into the strengths and limitations of different machine learning algorithms when applied to complex game environments. Through this project, I gained a solid understanding of the core concepts of reinforcement learning. I learned how to formulate a problem as a Markov Decision Process (MDP), identify the key components of an MDP, and apply reinforcement learning algorithms

to solve it. I also developed a strong intuition for the trade-off between exploration and exploitation, and how to balance these two competing objectives.

I gained practical experience with Q-learning and Deep Q-Learning, two fundamental reinforcement learning algorithms. I learned how to implement these algorithms from scratch, tune hyperparameters, and evaluate their performance. I also explored the advantages and limitations of each algorithm and understood when to use one over the other.

I learned the importance of feature engineering in reinforcement learning, especially when dealing with high-dimensional state spaces. I explored different feature extraction techniques and their impact on the agent's performance. I also learned how to evaluate the quality of features and select the most informative ones for the task at hand.

The aim of this research project is to contribute to the field of reinforcement learning by demonstrating its applicability to challenging game environments and providing insights into the development of intelligent agents capable of adapting to dynamic and uncertain situations. I believe the implementation codes and the statistic results would definitely contribute to the field of reinforcement learning. Throughout the project, it has been proven that Q-learning algorithm and its variations are effective methods to tackle complex game environment problems.

5.3 Improvement

There are several improvements can be made for this project. First and foremost, this project does not implement the Deep Q-Learning algorithm. There is code implementation for the Deep Q-Learning agent. However, it has been discovered that this Deep Q-Learning

agent takes much longer time than other Q-Learning algorithms. It requires more than 6000 episodes of training, with each training episode takes about 30 times longer than traditional Q-Learning agents. Overall, it takes about 100 times longer to train the Deep Q-Learning agent. This is because training deep neural networks can be computationally expensive, especially for large models and complex environments. This is due to the large number of parameters that need to be updated during training, as well as the need to perform backpropagation to calculate gradients. In addition, even the Deep Q-Learning agent is thoroughly trained to its maximum capacity, the trained agent fails most of the time at simple layouts like small grid. The reason behind it could be that the agent is poorly designed, or the code implementation is off from the design.

Another improvement can be made is the feature extraction function. The goal of feature extraction is to identify the most relevant information from the raw state representation and transform it into a more compact and informative representation that can be used by the learning agent. However, designing effective feature extractors can be challenging. In the experiments conducted with Approximate Q-Learning agent against classic Pacman layouts, the win rate is unstable across different runs, and the average win rate is below 90% for complex environment. Selecting the most relevant features from a large set of potential features can be challenging. Thus, more experiments should be conducted with the goal of selecting the most suitable features for this particular Pacman environment.

5.4 Overall

Overall, the Pacman project was a valuable learning experience. It provided me with a

hands-on introduction to reinforcement learning and helped me develop a strong foundation in this field. The result is somewhat close to my expectations at the beginning of the project. If I were to do it again, I would focus on studying more about the methodologies of previous studies on reinforcement learning. With a solid understanding of different variations of Q-Learning, I would be able to design and implement generally better agent with more cutting-edge techniques. On the other hand, exploring new aspects of existing algorithms is also important. I should also focus on improving the preexisting reinforcement learning algorithm.

6 Conclusion

This research project has explored the application of reinforcement learning algorithms to the classic Pacman game. By developing and evaluating reinforcement learning agents, I have gained valuable insights into the capabilities and limitations of reinforcement learning in complex, dynamic environments. The experiments demonstrated the effectiveness of Approximate Q-Learning in solving the Pacman game. The choice of feature extractor significantly influenced the agent's performance. The complexity of the layout had a noticeable impact on the agent's performance. And the results highlight the potential of reinforcement learning for solving challenging AI problems. The Deep Q-Learning algorithm is a comparatively new method compared to traditional Q-Learning algorithms. Therefore, it is worth exploring the field of deep reinforcing learning, demonstrates its potential and impact on the applications to real-world problems.

7 References

- [1] I. Szita and A. Lorincz, “Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations through Ms. Pac-Man,” *Journal of Artificial Intelligence Research*, vol. 30, pp. 659–684, Dec. 2007, doi: <https://doi.org/10.1613/jair.2368>.
- [2] M. H. Ang and O. Khatib, *Experimental Robotics IX*, vol. 21. Springer, 2006, pp. 363–372.
- [3] M. Wiering and Martijn Otterlo, *Reinforcement Learning State-of-the-Art*. Berlin Springer Berlin, 2014, pp. 579–610.
- [4] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, May 1992, doi: <https://doi.org/10.1007/bf00992698>.
- [5] G. Tesauro, “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” *Neural Computation*, vol. 6, no. 2, pp. 215–219, Mar. 1994, doi: <https://doi.org/10.1162/neco.1994.6.2.215>.
- [6] R. S. Sutton and A. Barto, *Reinforcement learning : an introduction*. Cambridge, Ma ; London: The Mit Press, 2018.
- [7] V. Mnih *et al.*, “Human-level Control through Deep Reinforcement Learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: <https://doi.org/10.1038/nature14236>.
- [8] M. Tamassia, F. Zambetta, W. L. Raffe, F. Mueller, and X. Li, “Learning Options From Demonstrations: A Pac-Man Case Study,” *IEEE Transactions on Games*, vol. 10, no. 1, pp. 91–96, Mar. 2018, doi: <https://doi.org/10.1109/tciaig.2017.2658659>.
- [9] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1–2, pp. 181–211, Aug. 1999, doi: [https://doi.org/10.1016/s0004-3702\(99\)00052-1](https://doi.org/10.1016/s0004-3702(99)00052-1).
- [10] P. Zang, P. Zhou, D. Minnen, and C. Isbell, “Discovering options from example trajectories,” Jun. 2009, doi: <https://doi.org/10.1145/1553374.1553529>.
- [11] G. Konidaris, S. Kuindersma, Roderic Grupen, and A. S. Barreto, “Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories,” *Neural Information Processing Systems*, vol. 23, pp. 1162–1170, Dec. 2010.
- [12] K. Subramanian, C. L. Isbell, and A. L. Thomaz, “Learning options through human interaction,” *IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT)*, 2011.

- [13] M. Hessel *et al.*, “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, doi: <https://doi.org/10.1609/aaai.v32i1.11796>.
- [14] A. Gnanasekaran, J. F. Faba, and J. An, “Reinforcement Learning in Pacman,” 2017, Available: <https://cs229.stanford.edu/proj2017/final-reports/5241109.pdf>
- [15] “Berkeley AI Materials,” *ai.berkeley.edu*. http://ai.berkeley.edu/project_overview.html