Final project report
Atari 2600 game (Alien)

Group 11

Belal El-hlwany **20398551**
Hassan Ahmed **20399136**
Manar El-ghobashy **20398569**

CISC 856
Reinforcement Learning

Supervised by

Dr. Sidney Givigi.
T.A. Yosra Kazemi.
T.A. Reem Abdel-salam.

1. Introduction

I. Project Overview

This project aims to train an AI agent to play the Atari 2600 game "Alien" using reinforcement learning techniques. The goal is to navigate through a maze-like environment, collect dots (destroy alien eggs), avoid enemies, and utilize power-ups to gain temporary advantages. The Atari 2600 game "Alien" serves as an ideal experiment due to its simplicity, well-defined rules, and clear objectives. By tackling this project, we can explore the capabilities of AI agents in solving complex real-world problems and gain insights into the application of deep reinforcement learning in-game environments.

II. Objectives

The primary objective of this project is to train an AI agent that can successfully play the Atari 2600 game "Alien" by effectively navigating the maze, collecting dots, avoiding enemies, and utilizing power-ups. The agent's performance will be evaluated based on its ability to accumulate high scores and complete levels efficiently.

III. Requirements/Task(s)

The task requires the AI agent to learn and master several gameplay mechanics within the "Alien" game. These mechanics include:

1. **Navigation:** The agent needs to navigate through a maze-like environment, similar to Pac-Man, while avoiding obstacles and enemies.
2. **Collecting Dots:** The agent must collect dots, which represent alien eggs in the game. Collecting these dots contributes to the agent's score and progress.
3. **Power-Ups:** The agent can collect power dots (pulsars) that provide temporary abilities to destroy enemies. Only one power dot is available at a time, and it appears in one of two predetermined locations upon collection.
4. **Enemy Interaction:** The agent needs to avoid contact with enemies roaming the maze. However, during the power-up phase, the agent can destroy enemies for a limited duration.
5. **Level Progression:** After successfully clearing a level, the agent engages in a bonus game. In this game, the agent must move upwards on the screen, similar to Freeway, and reach the prize at the top within a time constraint.

IV. challenges

1. **Complexity of the Game Environment:** The Atari 2600 game "Alien" presents a complex maze-like environment with multiple dynamic elements, such as enemies, power-ups, and changing levels. Navigating through the maze, avoiding enemies, and strategically using power-ups pose challenges for the AI agent.

2. **High-Dimensional State Space:** The state space in the Atari 2600 game "Alien" can be high-dimensional due to the pixel-based representation of the game screen. This makes it computationally expensive to process and learn from raw image data.

3. **Delayed Rewards:** The rewards in the game environment may be delayed, making it difficult for the agent to associate its actions with the resulting outcomes. The agent needs to learn to make long-term plans and anticipate future rewards.

4. **Optimization Challenges:** Training deep reinforcement learning models for complex game environments like "Alien" can be computationally expensive and time-consuming. Finding optimal hyperparameters, architecture choices, and training configurations becomes a challenge to ensure efficient learning and convergence.

5. **Generalization to New Levels:** Even after the agent successfully learns to play the game on a particular level, generalizing its knowledge to new levels can be challenging. Each level may introduce new obstacles, enemy placements, or time constraints, requiring the agent to adapt its learned strategies.

6. **Hyperparameter Tuning:** Choosing the right set of hyperparameters, such as learning rate, discount factor, exploration rate, and experience replay buffer size, plays a crucial role in the agent's performance. Finding the optimal hyperparameters requires careful tuning and experimentation.

7. **Overfitting:** The agent may face challenges related to overfitting, where it becomes too specialized to specific states or actions and fails to generalize well.

2. Problem Formulation

The problem of making a DQN agent solve the Pac-Man game can be formulated as follows:

1. **Environment:** The environment is represented by the Pac-Man game, which includes the game screen (RGB image) and the game state (e.g., Pac-Man's location, enemy locations, remaining lives, score, etc.). The agent interacts with the environment by selecting actions from the action space.

2. **State Space**: The state space consists of RGB images of the game screen with a shape of (210, 160, 3). Each image provides the agent with visual information about the game state, including the locations of Pac-Man, enemies, dots, power-ups, walls, and other relevant elements.

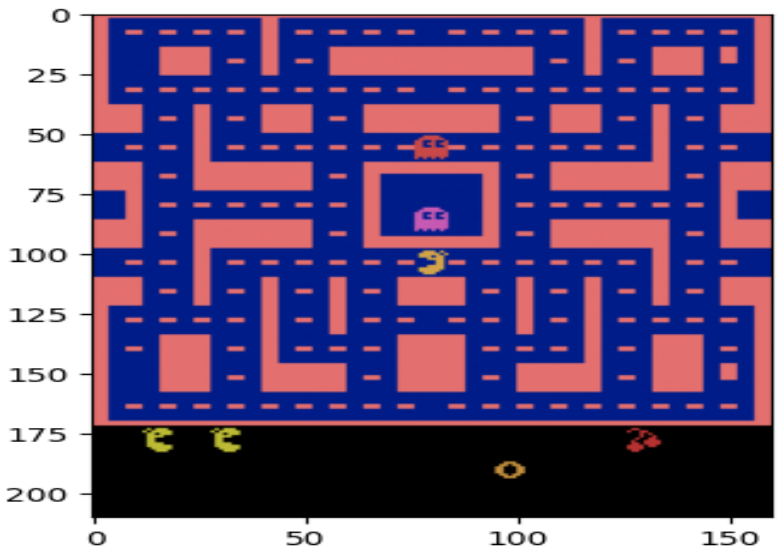

Figure 1 Pac-man Environment

3. **Action Space:** The action space comprises nine possible actions: move up, move down, move left, move right, upright, up left, down right, down left, fire, and do nothing. The agent selects actions based on the observed state to control Pac-Man's movement and actions in the game.

4. **Reward Scheme:** The reward scheme is designed to provide feedback to the agent based on its performance in the game. Positive rewards are given when the agent achieves objectives such as eating dots, power-ups, and advancing to the next level. Negative rewards are assigned for failing to achieve objectives, losing lives, or making mistakes that result in a loss of progress or points. The agent aims to maximize its cumulative reward over time by making strategic decisions.

5. **Objective:** The objective of the DQN agent is to learn a policy that maximizes its expected cumulative reward over multiple episodes of gameplay. The agent aims to navigate the maze, eat dots, collect power-ups, and avoid enemies (ghosts) while optimizing its score and progressing through the game levels.

6. **Deep Q-Network (DQN) Agent:** The DQN agent is trained using a combination of deep neural networks and reinforcement learning. It learns a Q-value function that estimates the expected future rewards for each state-action pair. The agent uses an epsilon-greedy policy to balance exploration and exploitation, gradually shifting from exploration to exploitation over time. The DQN agent learns from its experiences by replaying past experiences stored in a replay buffer and updating its Q-network using the Bellman equation.

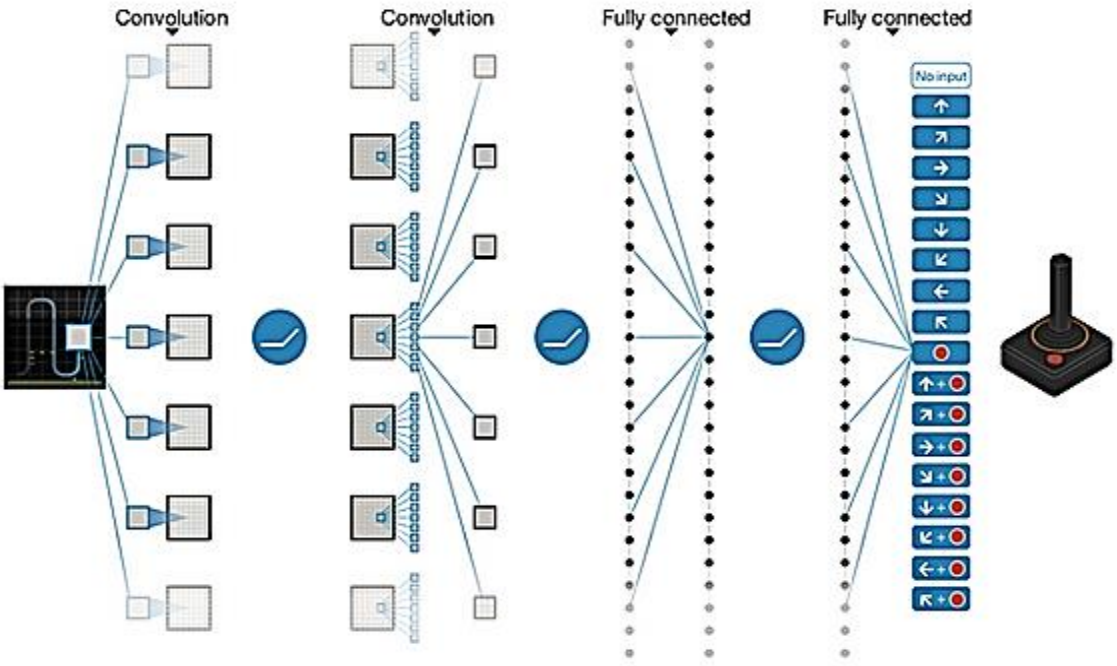

Figure 2 Schematic illustration of the convolutional neural network. The input to the neural network consists of an 84 * 84* 4 image produced by the preprocessing map w, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, max 0ð Þ, x)

3. Solution Overview (Implementation Details)

I. Preprocessing

When working with raw Atari 2600 frames, which have a high resolution of 210 x 160 pixels and a 128-color palette, it can be computationally and memory-intensive. To address this, a basic preprocessing step is applied to reduce the input dimensionality and handle certain artifacts specific to the Atari 2600 emulator.

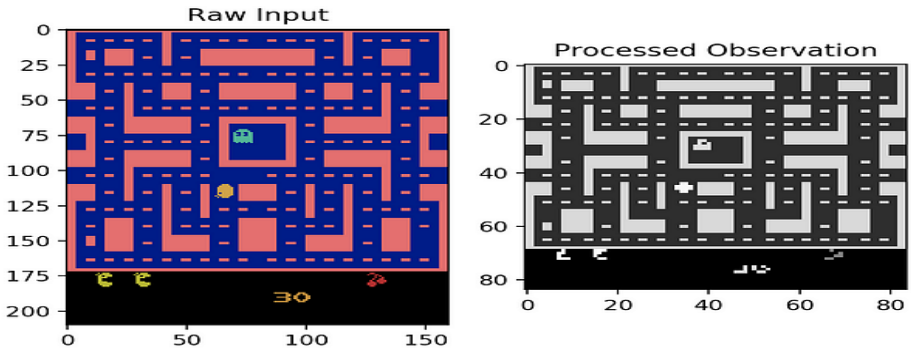

Figure 3 What we would see (left) vs. what our agent sees (right)

The luminance channel which represents the intensity or brightness of the image, is extracted from the RGB frame. This channel carries important visual information while reducing the complexity of the data. The extracted luminance channel is then resized to a smaller 84 x 84 dimension. This resizing further reduces the input dimensionality, making it more manageable for the neural network model to process. Additionally, to provide temporal context, the preprocessing step stacks the most recent m frames, typically chosen as m = 4, as the input to the Q-function. This allows the model to capture temporal dependencies and make informed decisions based on a sequence of frames. However, the algorithm is flexible and can accommodate different values of m, such as 3 or 5, without significantly impacting its performance.

Overall, these preprocessing steps help to alleviate the computational and memory requirements associated with working directly with raw Atari 2600 frames. By reducing the dimensionality, capturing temporal context, the processed frames serve as suitable inputs for the Q-function and enable effective training and decision-making in the reinforcement learning agent.

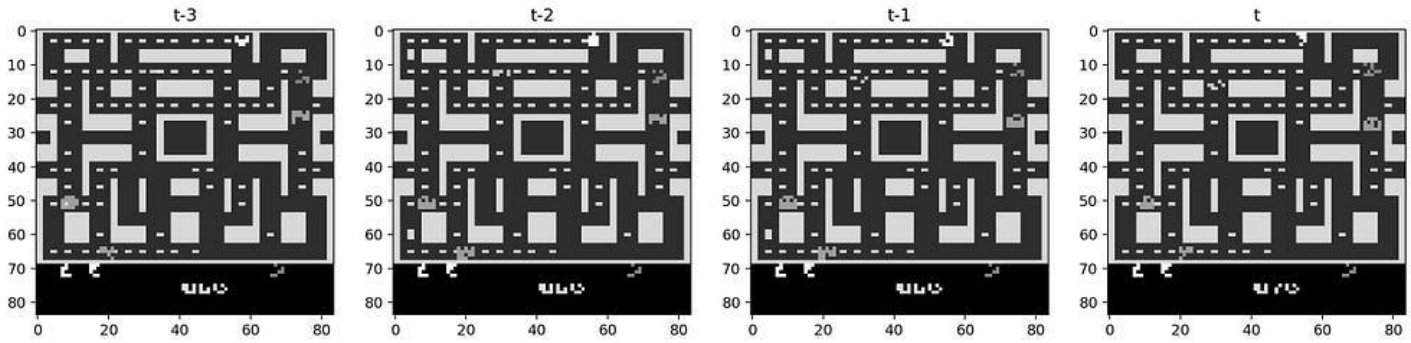

Figure 4 The stack of frames that makes up one complete input.

II. DQN Algorithm for Pac-Man

Our approach involves building a DQN agent with a window length for playing Pac-Man. The DQN algorithm, an extension of the Q-learning algorithm, is used to learn the optimal action-value function. The DQNAgentWithWindow class has various attributes such as state_size, action_size, learning_rate, gamma, epsilon, epsilon_min, epsilon_decay, batch_size, and memory.

The DQN agent utilizes a deep neural network architecture for function approximation. The build_model() function constructs a sequential neural network consisting of Conv2D layers for image processing, followed by a Flatten layer and fully connected (Dense) layers. The final layer uses a linear activation function to output the Q-values for each action. The model is compiled with mean squared error (MSE) loss and the Adam optimizer.

In the DQNAgentWithWindow class, the mapping from the environment state to the neural network input is done through a preprocessing step and the use of a window length.

The preprocessing step involves converting the raw image frames from the environment into grayscale images. The grayscale image is then resized to a smaller dimension of 84x84 pixels. This preprocessing step reduces the input dimensionality and simplifies the visual information for the neural network.

Next, the concept of a window length is introduced. The window length determines the number of consecutive frames that are stacked together to form a single input to the neural network. In this implementation, a window length of 4 is used, meaning that the agent considers the current frame along with the three previous frames as a single input.

To create the input tensor for the neural network, the preprocessed frames are stacked together along the first dimension, resulting in a tensor of shape (window_length, 84, 84, 1).
This stacked tensor represents the temporal context and allows the agent to capture the dynamics and motion information over multiple frames.

During training and evaluation, the stacked input tensor is fed into the neural network, and the model predicts the Q-values for each action. The action with the highest Q-value is chosen as the agent's action.

Note 1

Exploration-Exploitation Trade-off: The epsilon value controls the balance between exploration and exploitation in the agent's decision-making process. During training, a higher epsilon value encourages exploration, allowing the agent to explore different actions and states to discover optimal strategies. However, during evaluation, exploitation becomes more important, as the agent is expected to use its learned knowledge to make optimal decisions. Setting epsilon to a small value (0.05 in this case) ensures a higher focus on exploitation, as the agent predominantly selects actions based on its learned policy.

We know that, it's better to set the value of epsilon to zero but the training episode is not enough. So, setting the value of epsilon to 0.05 during the evaluation process allows the agent to primarily exploit its learned knowledge and assess its performance based on its acquired policy. This approach aligns with the exploration-exploitation trade-off and provides a more accurate evaluation of the agent's abilities in making optimal decisions.

Note 2

To accelerate the training process, we make the agent has one live only because when we tried the first version of the game with three lives the time for one episode was very long.

III. Deep Q-Network" (DQN) architecture

The neural network used in the DQN algorithm typically consists of convolutional layers followed by dense layers. The convolutional layers are used to extract features from the input state, while the dense layers are used to map the features to the Q-values for each action. The neural network takes in an image of size (84, 84, 1) and outputs the Q-values for each action. The loss function used is Mean Squared Error (MSE), and the optimizer used is Adam.

Table 1 : The neural network architecture.

Architecture		
Layer	Layer size	Activation
Conv2d	32	Relu
Conv2d	64	Relu
Conv2d	64	Relu
Dense	512	Relu
Dense	Action_size= 9	Linear
Compile		
Loss	MSE	
Optimizer	Adam	

IV. Techniques used in the training process

1. The DQN with replay buffer
2. The DQN with replay buffer and warmup episodes
3. The DQN with replay buffer, warmup, and window size

The second and third techniques are used to improve the agent training process. One is the addition of warm-up, and the other is the window technique. The agent with warmup technique is a type of reinforcement learning algorithm that gradually introduces an agent to an environment by starting with a period of random actions before transitioning to using the agent's learned policy. The algorithm interacts with the environment for a specified number of episodes, taking random actions during the warm-up period to explore and learn about the environment. After the warm-up period, the agent switches to using its learned policy to take actions, with some probability of taking a random action based on the exploration rate. The observations, actions, rewards, and next state are stored in a replay buffer, which is used to train the neural network model during the learning phase. This technique can help the agent to avoid getting stuck in suboptimal policies and improve the overall performance of the algorithm.

The agent with window technique is a variation of the Deep Q-Network (DQN) algorithm that uses a window of previous states as input to the neural network. This technique allows the agent to capture temporal dependencies in the environment and make decisions based on past observations. During training, the agent stores experiences in a replay buffer and trains the neural network to minimize the mean squared error between the predicted Q-values and the target Q-values. During testing, the agent uses the same neural network model with a window of previous states as input to select actions. This technique can be further improved by using techniques such as prioritized experience replays, double Q-learning, and dueling networks.

V. Environment

We used MSpacman-4 and deterministic MSpacman-4 and the difference between them is:

In MSpacman-4, the environment follows the original Ms. Pac-Man game rules where the ghosts' movements are stochastic or random. The ghosts' behavior is influenced by a combination of predefined patterns and random actions, making their movements less predictable. This randomness adds an additional challenge for the agent, as it needs to adapt to the changing ghost behavior and make decisions accordingly.

On the other hand, deterministic MSpacman-4 eliminates the stochasticity in the ghosts' movements. The ghosts now follow predetermined patterns and behave in a predictable manner. This deterministic behavior makes it easier for the agent to learn and plan its actions, as it can anticipate the ghosts' movements and formulate optimal strategies accordingly.

4. Results

We manually performed hyperparameter tuning without employing any search algorithms. We conducted several trials, each involving a different combination of hyperparameters, to evaluate the performance of the algorithm under varying conditions. As we mentioned before, we applied three different techniques and performed five different trials on them to be able to compare the performance of the agent with different techniques and hyperparameters.

A. For the DQN with replay buffer

- **The hyperparameters for this trial are:**

Table 2: Hyperparameters used in trial one.

episodes=100	learning_rate=0.01
gamma=0.9	epsilon=.9
epsilon_min=0.1	epsilon_decay=0.995
batch_size=32	memory_size=10000

Training Curves:

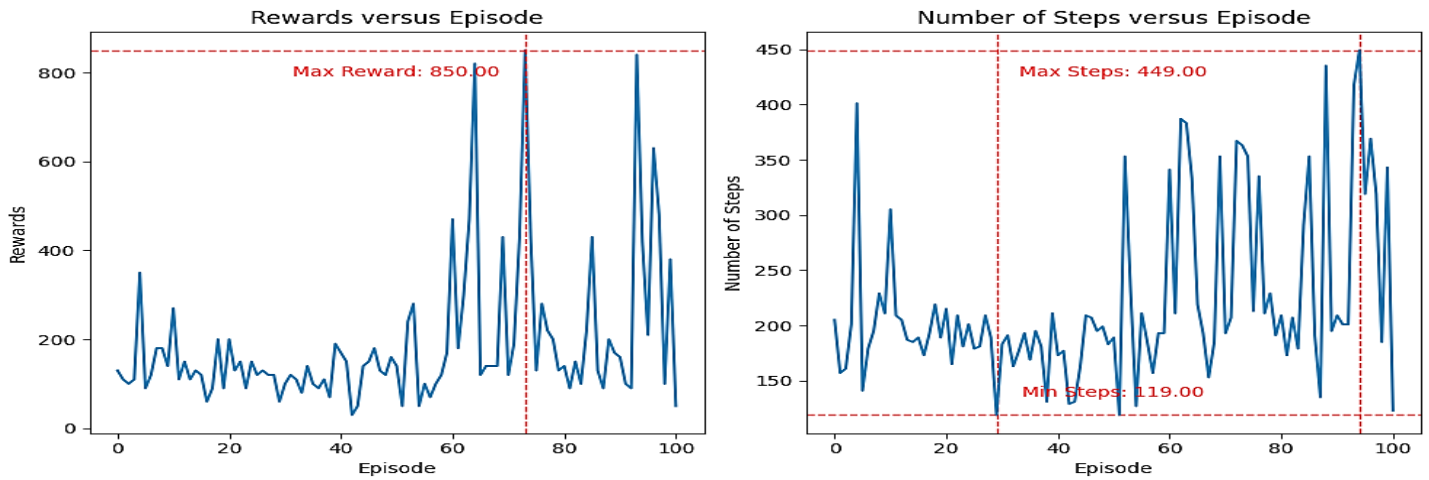

Figure 5: Training curves of the first trial.

Evaluations Curves:

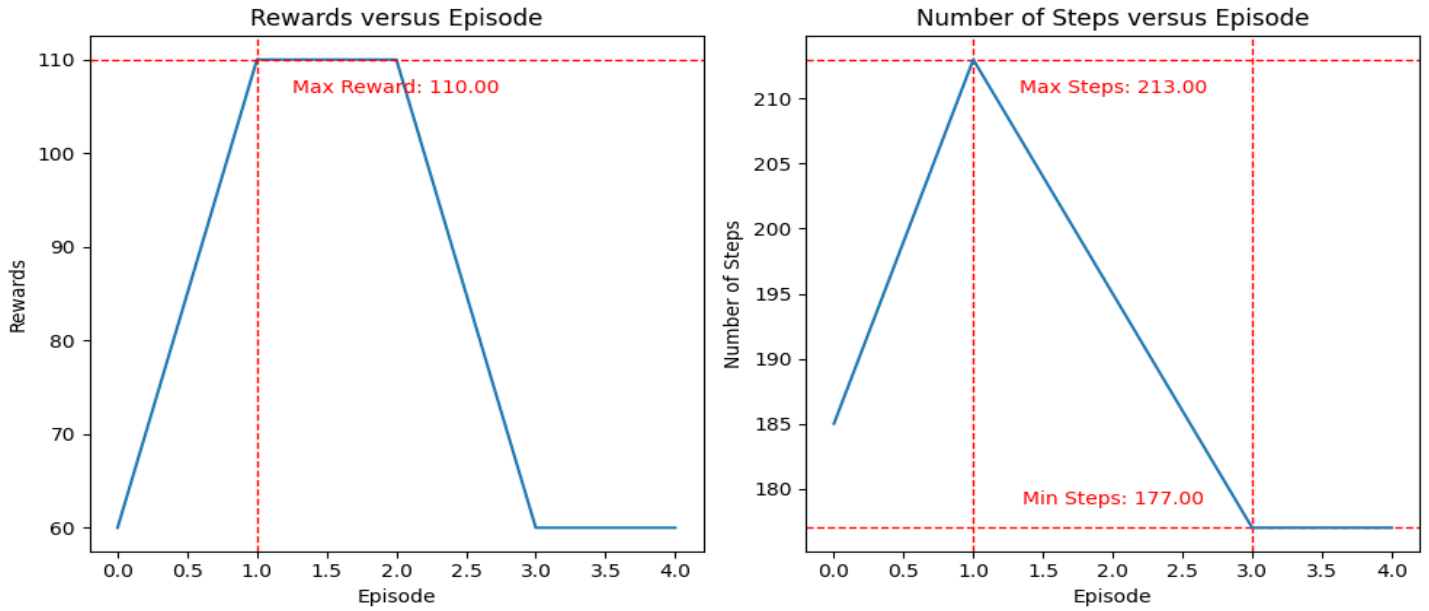

Figure 6: Evaluation curves of the first trial.

Observations

- The number of episodes is not enough for the agent to learn
- There is high oscillation in the reward and timestep per each episode due to the high complexity of the observation space.
- In training, The DQN with replay buffer (100 episodes):
    - Mean reward over the last 10 episodes: 328.00
    - Std dev of rewards over the last 10 episodes: 221.94
    - Minimum reward in those episodes: 50.00

    - Maximum reward in those episodes: 630.00
- In evaluation, The DQN with replay buffer (5 episodes):
    - Average time steps over 5 testing episodes: 189.4
    - Average score over 5 testing episodes: 80.0
    - Rewards for each testing episode: [60.0, 110.0, 110.0, 60.0, 60.0]
    - Standard deviation of rewards: 24.49489742783178
    - The average score over the 5 testing episodes is 80.0, with individual episode scores ranging from 60.0 to 110.0. The standard deviation of rewards is 24.5.
- The performance of the DQN with replay buffer appears to be lower than initially stated.
- The mean reward and average time steps remain the same, but the average score over testing episodes has decreased.
- Additionally, the standard deviation of rewards indicates a moderate variation in performance across episodes.

B. The DQN with replay buffer and warmup episodes

- **The hyperparameters for this trial are:**

Table 3: Hyperparameters used in trial 2.

episodes=150 (warmup = 50) (agent = 100)	learning_rate=0.001
gamma=0.9	epsilon=.9
epsilon_min=0.1	epsilon_decay=0.995
batch_size=64	memory_size=10000

Training Curves:

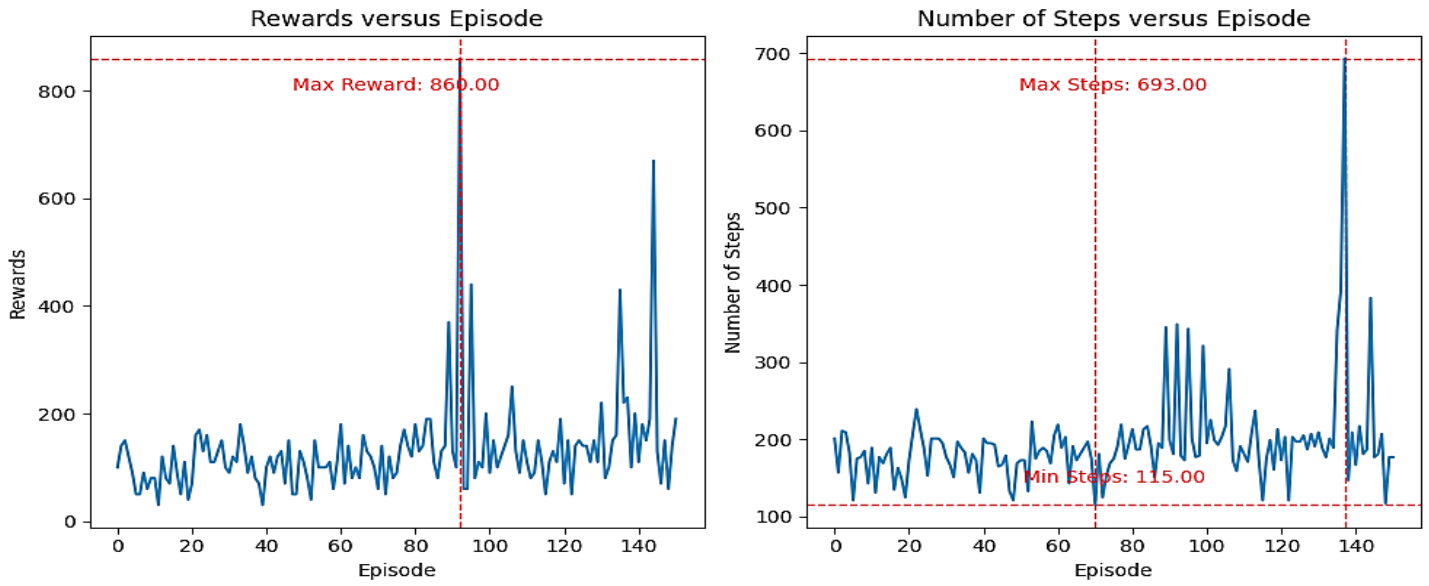

Figure 7: Training curves of the second trial.

Evaluations Curves:

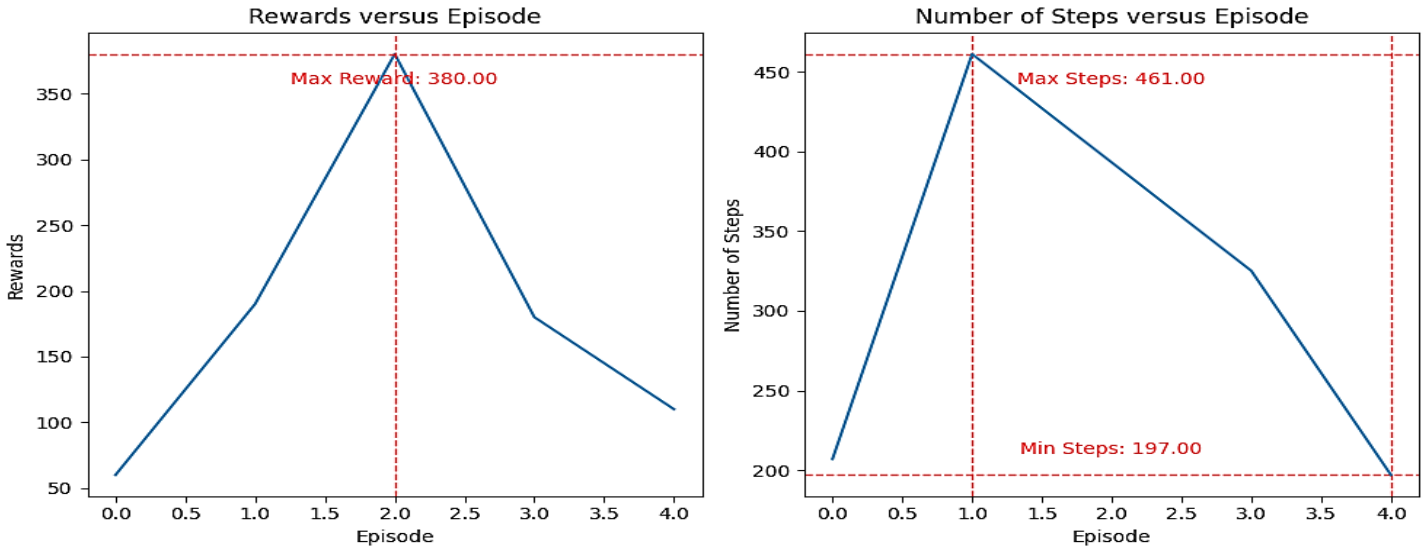

Figure 8: Evaluation curves of the second trail.

Observations

- The number of episodes is not enough for the agent to learn
- The stability increased after adding the warmup episodes except for the episode ~ 93
- In training, DQN with replay buffer and warmup episodes (150 episodes):
    - Mean reward over the last 10 episodes: 193.00
    - Std dev of rewards over the last 10 episodes: 164.62
    - Minimum reward in those episodes: 60.00
    - Maximum reward in those episodes: 670.00
- In evaluation, DQN with replay buffer and warmup episodes (5 episodes):
    - Average time steps over 5 testing episodes: 316.6
    - Average score over 5 testing episodes: 184.0
    - Rewards for each testing episode: [60.0, 190.0, 380.0, 180.0, 110.0]
    - Standard deviation of rewards: 109
    - The average score over the 5 testing episodes is 184.0, with individual episode scores ranging from 60.0 to 380.0. The standard deviation of rewards is 109.
- The DQN with replay buffer and warmup episodes shows a higher average score over the testing episodes, indicating better performance.
- The standard deviation of rewards is also higher, suggesting a larger variation in episode scores.
- It's recommended to thoroughly evaluate different approaches (with high number of episodes) and potentially experiment with other variations to determine the most suitable solution for your specific needs.

C. The DQN with replay buffer, warmup and window size

- **The hyperparameters for this the first two trials are:**

Table 4: Hyperparameters used in trials 3 & 4.

Trial (1)	Trial (2)
episodes=200 (warmup = 70) (agent = 130)	episodes=200 (warmup = 50) (agent = 150)
gamma=0.9	gamma=0.9
epsilon_min=0.1	epsilon_min=0.1
batch_size=64	batch_size=32
learning_rate=0.001	learning_rate=0.01
epsilon=1	epsilon=0.9
epsilon_decay=0.995	epsilon_decay=0.995
memory_size=10000	memory_size=10000

Training Curves Trial (1)

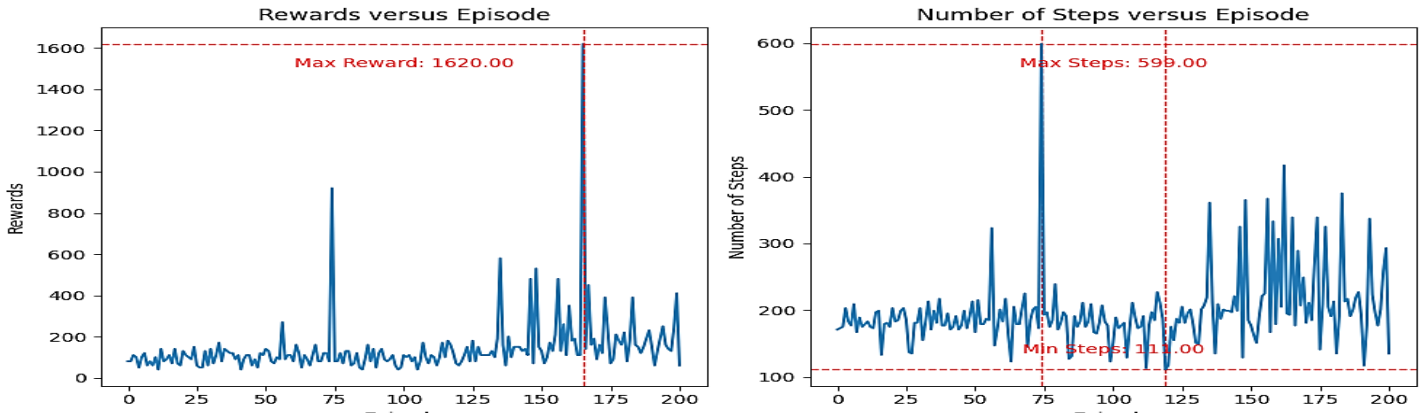

Figure 9: Training curves of trial (1) in C.

Training Curves Trial (2)

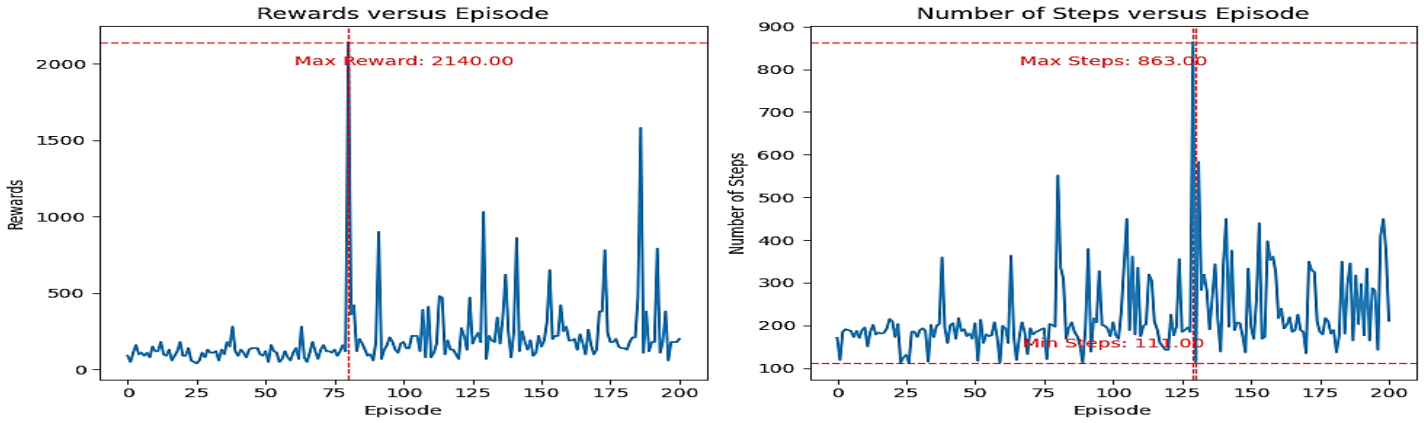

Figure 10: Training curve of the trial (2) in C.

Evaluations Curves Trial (1):

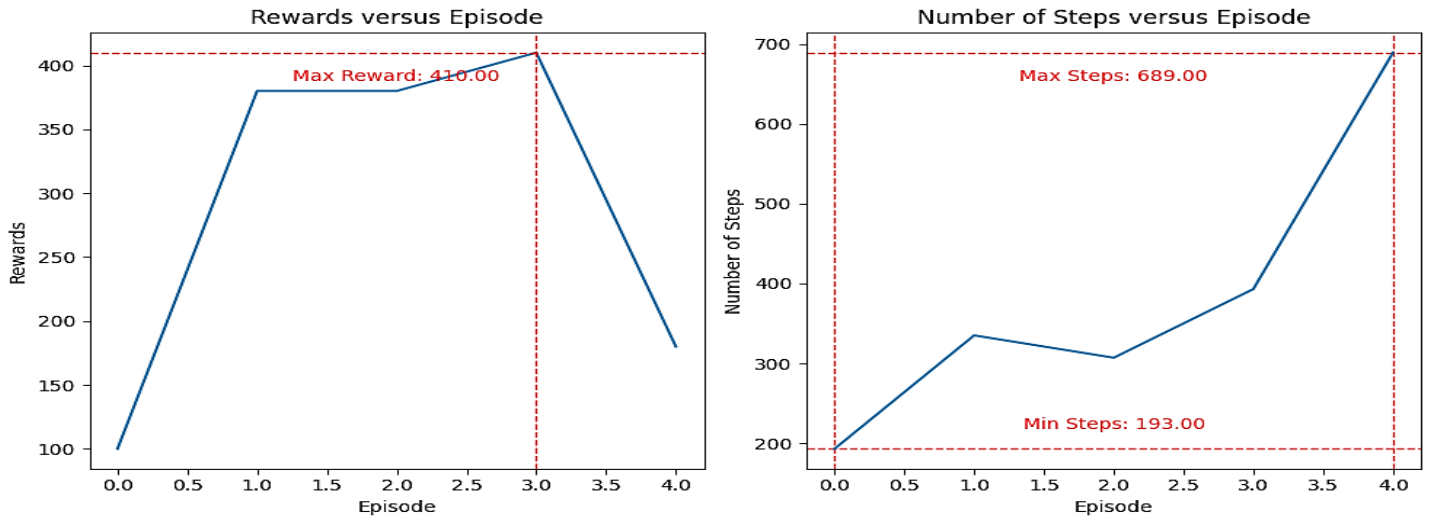

Figure 11: Evaluation curve of the trail (1) in C.

Evaluations Curves Trial (2):

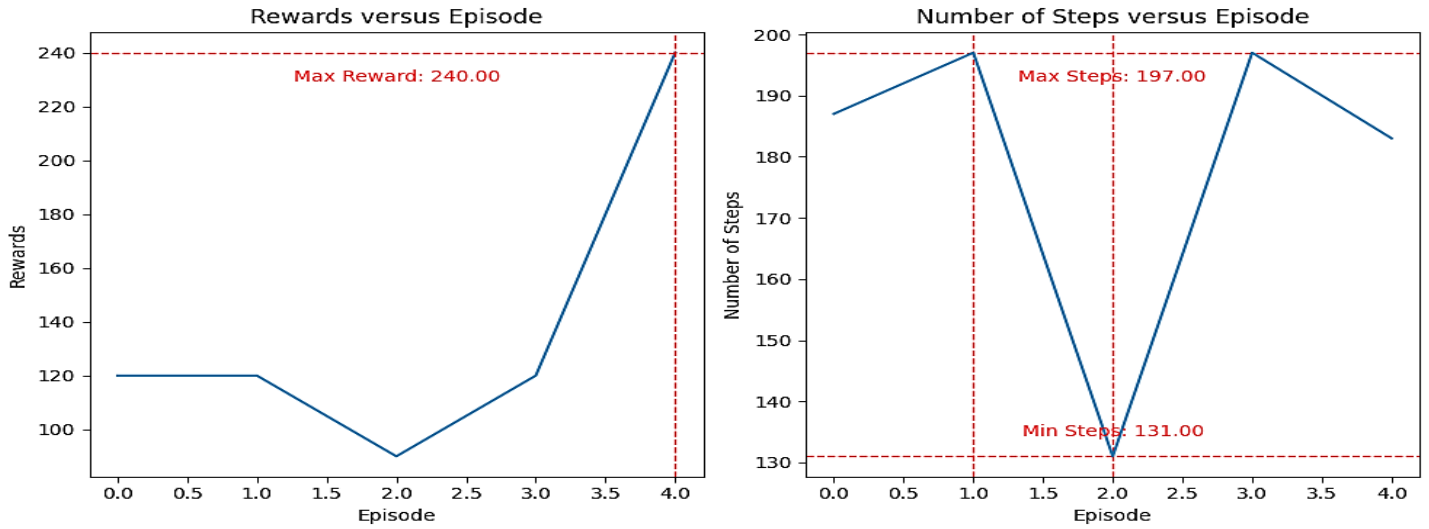

Figure 12: Evaluation curves of the trail (2) in C.

Observations

Table 5: Observations of the last two trials.

Trial (1)	Trial (2)
Mean reward over last 10 episodes: 177.00	Mean reward over last 10 episodes: 244.00
Std dev of rewards over last 10 episodes: 98.60	Std dev of rewards over last 10 episodes: 197.70
Minimum reward in those episodes: 60.00	Minimum reward in those episodes: 60.00
Maximum reward in those episodes: 410.00	Maximum reward in those episodes: 790.00

- In training, DQN with replay buffer and warmup episodes (150 episodes)
    - The reward in the trial (2) higher than the reward in the trial (1) or in another word the oscillation of the trial (2) is higher than trial (1).
    - The reason of that is the batch size of the trial (2) is 32 compared with batch size in the trial (1) which is 64 (when the batch size increase, this leads to more stable curve).

Note

- In the next trial we fulfill the requirements for the training criteria after every 10 episodes of training run the estimated policy in the environment for 5 test episodes.
- We haven't done this for the other trials because the training time was long and we don't have enough resources.
- We also make the agent interact with the environment for one live only in all the trials to accelerate the training process.

The DQN with replay buffer, warmup and window size (Last Trial)

- **The hyperparameters for the last trial are:**

Table 6: Hyperparameters used in the last trial.

episodes=300 (warmup = 200) (agent = 100)	learning_rate=0.001
gamma=0.9	epsilon=.9
epsilon_min=0.1	epsilon_decay=0.995
batch_size=32	memory_size=100000

Sample of the training output:

```
Episode 250: Mean reward over last 10 episodes: 171.00   , Std dev of rewards over last 10 episodes: 56.47   ,epsilon:0.29   ,min. reward in those episode :110.00   ,max in those episode: 330.00
Average score over 5 testing episodes:  132.0  and the scores = [110.0, 180.0, 100.0, 200.0, 70.0]
std of rewards 49.558046773455466
Average time steps over 5 testing episodes:  199.0
Episode 260: Mean reward over last 10 episodes: 194.00   , Std dev of rewards over last 10 episodes: 79.77   ,epsilon:0.27   ,min. reward in those episode :90.00   ,max in those episode: 320.00
Average score over 5 testing episodes:  120.0  and the scores = [160.0, 90.0, 230.0, 60.0, 60.0]
std of rewards 66.03029607687671
Average time steps over 5 testing episodes:  197.8
Episode 270: Mean reward over last 10 episodes: 214.00   , Std dev of rewards over last 10 episodes: 127.61   ,epsilon:0.26   ,min. reward in those episode :90.00   ,max in those episode: 480.00
Average score over 5 testing episodes:  154.0  and the scores = [70.0, 80.0, 100.0, 440.0, 80.0]
std of rewards 143.33178293735133
Average time steps over 5 testing episodes:  213.4
Episode 280: Mean reward over last 10 episodes: 401.00   , Std dev of rewards over last 10 episodes: 451.27   ,epsilon:0.25   ,min. reward in those episode :70.00   ,max in those episode: 1710.00
Average score over 5 testing episodes:  362.0  and the scores = [240.0, 850.0, 150.0, 100.0, 470.0]
std of rewards 275.056357861439
Average time steps over 5 testing episodes:  278.6
Episode 290: Mean reward over last 10 episodes: 207.00   , Std dev of rewards over last 10 episodes: 67.98   ,epsilon:0.23   ,min. reward in those episode :100.00   ,max in those episode: 350.00
Average score over 5 testing episodes:  232.0  and the scores = [80.0, 440.0, 80.0, 450.0, 110.0]
std of rewards 174.28711943227475
Average time steps over 5 testing episodes:  255.4
Episode 300: Mean reward over last 10 episodes: 281.00   , Std dev of rewards over last 10 episodes: 96.90   ,epsilon:0.22   ,min. reward in those episode :160.00   ,max in those episode: 440.00
Average score over 5 testing episodes:  166.0  and the scores = [70.0, 70.0, 210.0, 410.0, 70.0]
std of rewards 133.5065541462291
Average time steps over 5 testing episodes:  286.2
```

Figure 13: Sample of the training output in the last trial.

Training Curves:

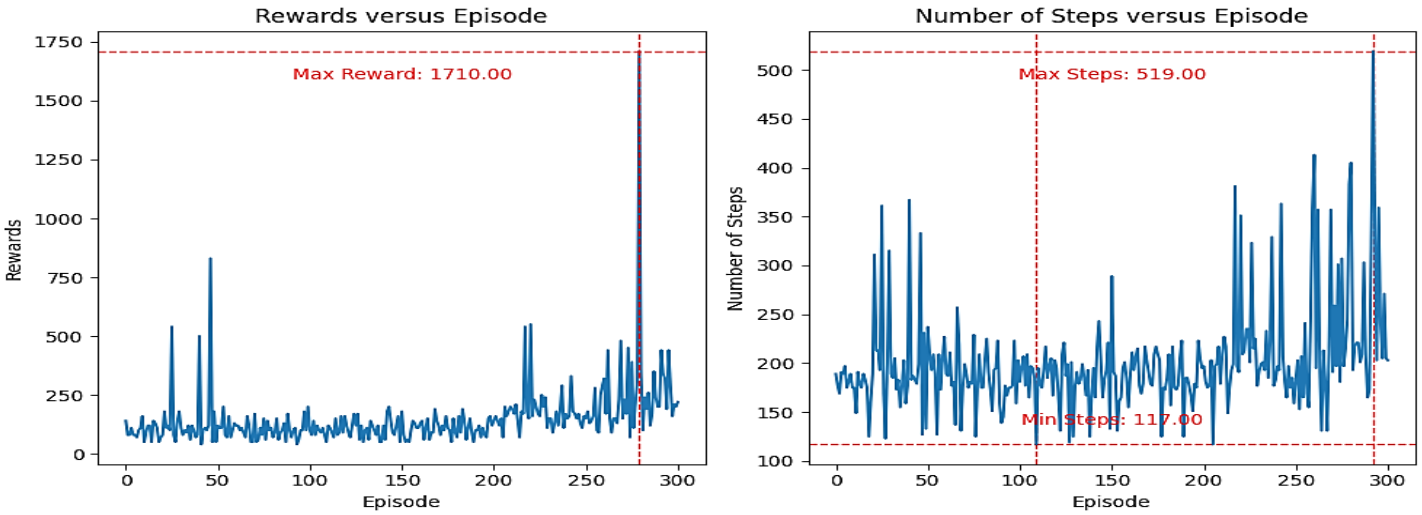

Figure 14: Training curve of the last trial.

Evaluation every 10 episodes of training:

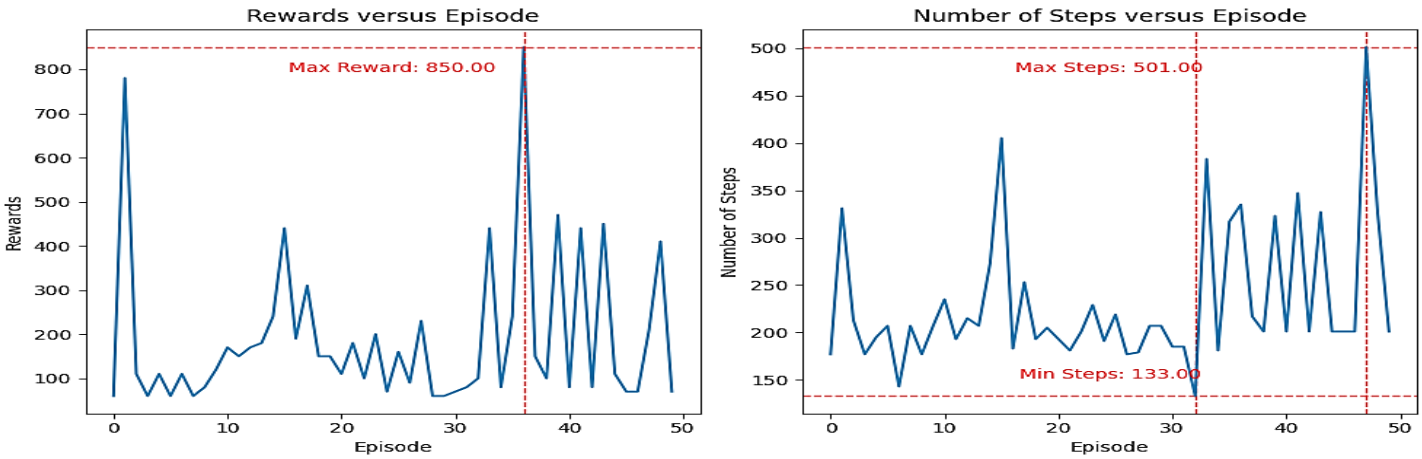

Figure 15: Evaluation curve for every ten episodes of training in the last trial.

Evaluation Curves:

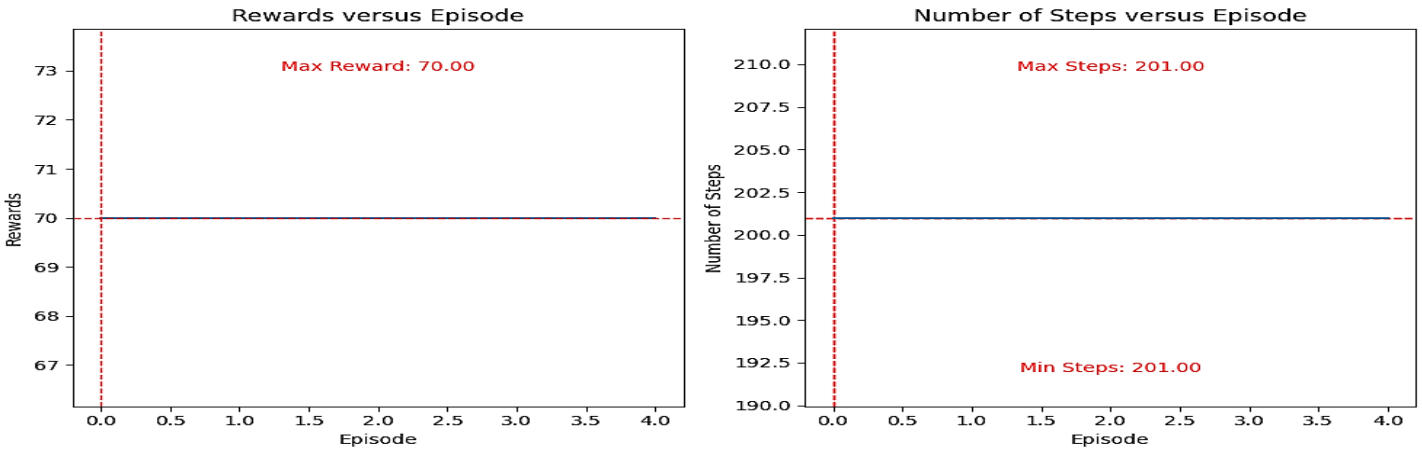

Figure 16: Evaluation curves of the last trials.

- If we set the epsilon during the evaluation to zero as we can see the reward is constant which means that the agent can't generalize the state space
- If we increase the epsilon to 0.05 during the evaluation, the result will be as follow:

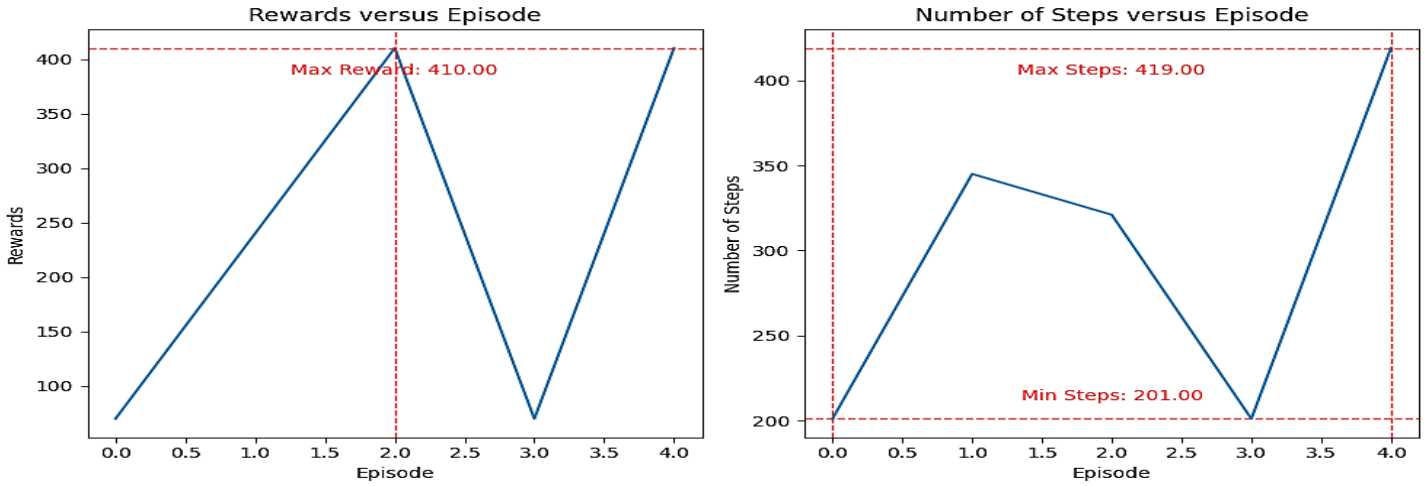

Figure 17: Evaluating curve with epsilon equal to 0.05 in the last trial.

- And also, when evaluating the environment with three lives the result will be as follow:

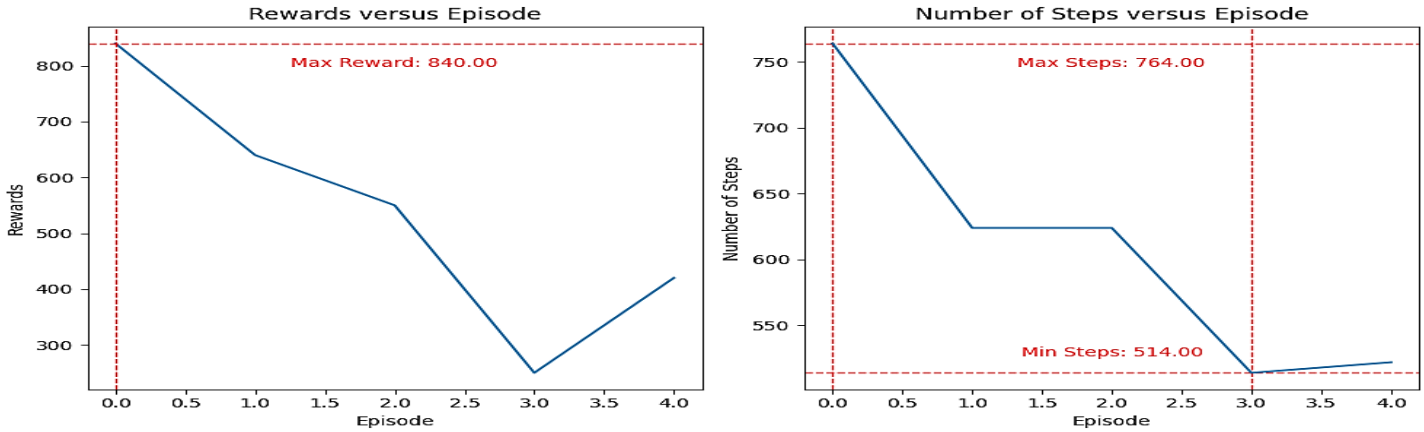

Figure 18: Evaluating curve with three lives.

Observations

- DQN with replay buffer, warmup episodes, and window size (300 episodes):
  - Mean reward over the last 10 episodes: Ranges from 92 to 401.
  - Std dev of rewards over the last 10 episodes: Ranges from 28.57 to 451.27.
  - The minimum and maximum rewards achieved in those episodes vary.
  - The epsilon value decreases gradually from 0.95 to 0.22.
- The agent displays improvement in terms of mean rewards over the training episodes.
- Based on these plots, The DQN with replay buffer, warmup episodes, and window size shows the most promising results in terms of achieving higher mean rewards, although it may exhibit greater variability in performance.

5. Conclusion

- In this project, we trained a reinforcement learning agent to play Pac-Man using the DQN algorithm with various techniques such as replay buffer, warmup episodes, and window size. We conducted several trials with different hyperparameter settings and evaluated the agent's performance under varying conditions.
- Through this project, we learned the importance of hyperparameter tuning, appropriate technique selection, thorough evaluation, and experimentation to achieve the best results. We observed that the addition of warmup episodes and window size helped to improve the agent's performance, and the best-performing trial was the DQN with replay buffer, warmup episodes, and window size.
- We can conclude that all three scenarios show progress in terms of mean rewards over the training episodes. However, the DQN with replay buffer, warmup episodes, and window size (scenario 3) achieves the highest mean reward of 401. It also exhibits the highest standard deviation of rewards, indicating a greater variance in performance. This suggests that the agent's performance may be more unstable compared to the other scenarios.
- (100,150,200,300) episodes are not enough to train the agent since we need approximately 12000 episodes to converge and we don't have the required resources for this.
- We get this number (12000 episodes) from the related work as mentioned in this link : Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning | by Jake Grigsby | Towards Data Science
- Two videos showcasing the agent's gameplay in Pac-Man of the last trial have been uploaded and are accessible via the hyperlink provided below: https://drive.google.com/drive/folders/1ir9eLQPCQ6WmvwAJNLHL1nVOehrQYypt?usp=sharing
- Overall, this project provided valuable insights into training a reinforcement learning agent and the challenges involved in achieving good performance in complex environments.