Reinforcement Learning – Final Course Project

Erez Soffer (ID. 209648682)

Submitted as final project report for the RL course, RUNI, 2024

1 Introduction

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by interacting with an environment. The primary goal is to develop a system that can learn the best actions to take in various situations to achieve a specific objective.

In this project, we focus on applying RL to a classic puzzle game known as Sokoban. Sokoban is a puzzle game where the player pushes boxes to designated storage locations within a maze-like environment. The challenge is to find an efficient strategy to complete the puzzle, making it an ideal candidate for RL.

1.1 Motivation

The primary motivation behind this project is to explore how RL algorithms can be used to solve complex decision-making tasks. Sokoban, with its intricate rules and spatial constraints, offers a challenging environment for testing RL techniques. By training an RL agent to solve Sokoban puzzles, we can gain insights into how well RL algorithms can handle real-world problems that involve spatial reasoning and strategic planning.

1.2 Environment Analysis

- **State Space:** In Sokoban, the state of the environment is defined by the positions of the player and the boxes within the maze. Each unique arrangement of these elements represents a different state. The state space can be large and complex due to the various configurations possible in the maze.
- **Action Space:** The agent can perform actions such as moving up, down, left, or right. Each action affects the position of the player and the boxes in the environment.
- **Rewards:** The primary reward in Sokoban is given when a box is successfully pushed to its target location. The agent receives positive rewards for completing goals and penalties for making inefficient moves or failing to make progress.

In summary, this project aims to use RL to develop an agent that can efficiently solve Sokoban puzzles. We will analyze how different RL techniques perform in this environment and evaluate the effectiveness of our approach in learning and executing the optimal strategies for solving the puzzles.

2 Solution

2.1 General Approach

To solve the Sokoban puzzle using Reinforcement Learning (RL), we adopted a deep Q-learning approach, specifically employing a Deep Q-network (DQN). This method allows the agent to learn from its interactions with the environment by approximating the Q-value function through a neural network.

The DQN approach was selected for its effectiveness in handling large state spaces and complex decision-making problems, such as those presented in Sokoban. The core idea of DQN is to use a neural network to estimate the Q-values for each action, which helps in making informed decisions based on the current state of the environment.

2.1.1 Alternative Approaches

While DQN is a robust method, several alternatives were considered:

- **Monte Carlo Methods:** These methods could be used for estimating value functions based on average returns. However, they may not be as efficient in environments with large state spaces or where exploration is critical.
- **Policy Gradient Methods:** These methods focus on optimizing the policy directly rather than the Q-value. Techniques like Actor-Critic could be explored, but they might require more complex implementations and tuning.
- **Actor-Critic Methods:** These combine the benefits of value-based and policy-based methods. They are suitable for environments with continuous action spaces but may be more complex to implement.

Each alternative has its advantages and trade-offs. DQN was chosen for its balance between complexity and performance, particularly for environments with discrete actions and large state spaces.

2.2 Design

The solution was implemented using the following components:

2.2.1 Platform

The project was developed and tested using Python with TensorFlow and Keras libraries. The code was executed on Google Colab with an NVIDIA GPU to accelerate training.

2.2.2 Training Duration

Training the DQN agent took approximately 2 hours for 1000 episodes. The training time varied based on the complexity of the Sokoban puzzles and the computational resources available.

2.2.3 Technical Challenges

- **State Representation:** Converting the Sokoban environment into a format suitable for the neural network involved preprocessing the observations to grayscale images and resizing them.
- **Exploration vs. Exploitation:** Balancing exploration and exploitation was crucial for ensuring the agent could explore new strategies while refining known ones.
- **Reward Shaping:** Designing the reward function to effectively encourage the agent to solve puzzles efficiently was challenging.

2.2.4 Loss Functions and Optimizers

The neural network used a Mean Squared Error (MSE) loss function, which is standard for regression tasks in Q-learning. The Adam optimizer was chosen due to its effectiveness in training deep neural networks, providing adaptive learning rates and good convergence properties.

2.2.5 Architecture

The DQN model employed a convolutional neural network (CNN) with the following architecture:

- **Convolutional Layers:** Three convolutional layers with increasing filter sizes and strides were used to extract features from the input images.
- **Fully Connected Layers:** Several dense layers followed the convolutional layers to learn high-level representations and predict Q-values for each action.
- **Output Layer:** The output layer had neurons equal to the number of possible actions, with linear activation to estimate the Q-values.

The target model was periodically updated to improve training stability and convergence.

2.2.6 Additional Considerations

To enhance training, techniques such as experience replay and prioritized replay were implemented. Experience replay helps in breaking the correlation between consecutive samples by storing and randomly sampling past experiences. Prioritized replay, on the other hand, allows the agent to sample more important experiences more frequently, improving learning efficiency.

3 Experimental Results

3.1 Ex1: Double Q-Learning with Prioritized Experience Replay

Objective: Evaluate the performance of a Double Q-Learning agent integrated with Prioritized Experience Replay on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 500 steps max)
- **Agent:** Double Q-Learning with Prioritized Experience Replay
- **Hyperparameters:**
  - Learning Rate: 0.1
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0
  - Memory Size: 10,000 transitions
  - Batch Size: 64
  - Priority Exponent (α): 0.6
  - Importance Sampling Exponent (β): 0.4

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 1: Results for Ex1: Double Q-Learning with Prioritized Experience Replay

Episode	Reward	Avg Reward	Std Dev	Steps
100	5.30	-42.08	16.01	57
200	-10.60	-26.85	23.16	216
300	10.00	0.52	20.93	10
354	10.10	10.08	0.08	9
355	10.10	10.08	0.08	9
356	10.10	10.08	0.08	9
357	10.10	10.09	0.04	9
358	10.10	10.09	0.04	9

Figures:

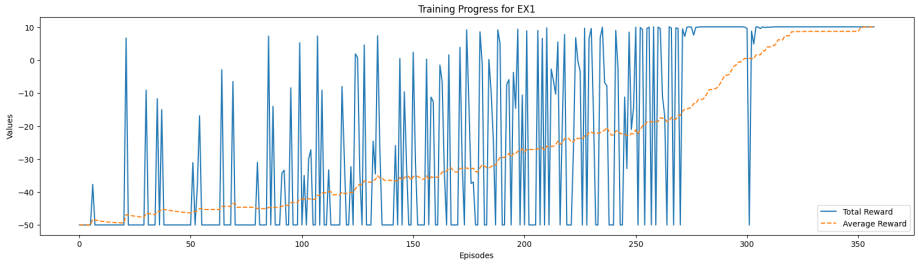

Figure 1: Training Progress for EX1

3.2 Ex2: "Push & Pull" – RANDOM GENERATED SCENARIO

In Ex2 we perform 7 experiments:

3.2.1 First Experiment

Objective: Evaluate the performance of a Deep Q-Network (DQN) agent on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 500 steps max)
- **Agent:** Deep Q-Network (DQN)
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.3
  - Memory Size: 2,000 transitions
  - Batch Size: 64

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 2: Results for the First Experiment: Deep Q-Network

Episode	Reward	Avg Reward	Std Dev	Steps
100	-50.00	-34.14	22.41	500
200	-50.00	-35.62	22.97	500
300	-50.00	-33.10	25.25	500
400	-34.60	-40.45	19.55	456
500	-50.00	-37.54	21.93	500
600	-50.00	-37.94	22.71	500
700	-50.00	-38.96	20.60	500
800	-50.00	-41.04	19.65	500
900	-50.00	-44.54	15.12	500
1000	-50.00	-44.37	15.69	500

Figures:

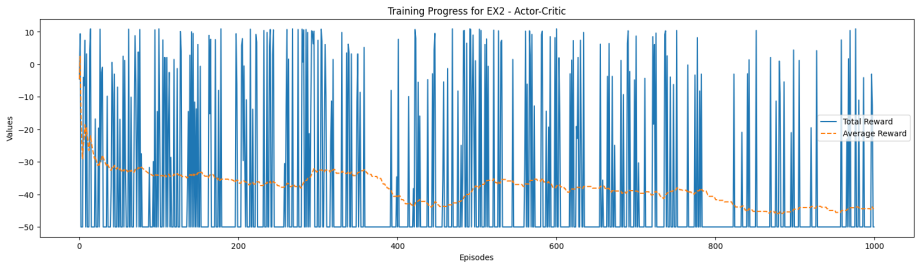

Figure 2: Training Progress for the First Experiment

3.2.2 Second Experiment

Objective: Assess the performance of the DQN agent with varying batch sizes and success thresholds on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Deep Q-Network (DQN)
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.3

- Memory Size: 2,000 transitions
- Batch Size: 8
- Success Threshold: -2

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 3: Results for the Second Experiment: DQN with Batch Size 8 and Success Threshold -2

Episode	Reward	Avg Reward	Std Dev	Steps
500	-2.00	-0.16	4.38	20
1000	-2.00	-0.91	3.48	20
1500	-2.00	-1.17	3.04	20
2000	-2.00	-1.77	1.64	20
2500	-2.00	0.42	5.00	20
3000	-2.00	0.94	5.38	20
3500	-2.00	-1.75	1.72	20
4000	-2.00	-1.62	2.17	20
4500	-2.00	-0.88	3.57	20
5000	-2.00	-0.59	4.00	20

Figures:

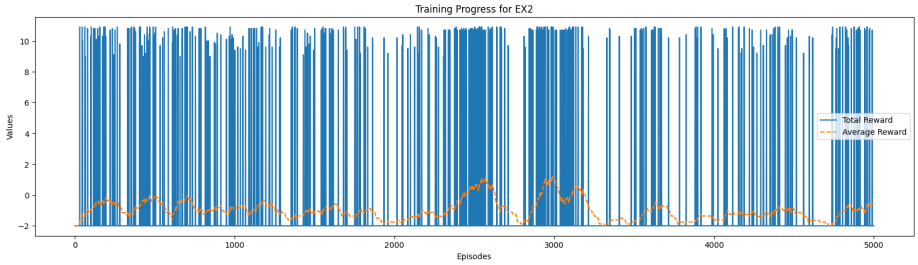

Figure 3: Training Progress for the Second Experiment

3.2.3 Third Experiment

Objective: Examine the impact of more frequent updates in the DQN agent on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Deep Q-Network (DQN)
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.3
  - Memory Size: 2,000 transitions
  - Batch Size: 8
  - Success Threshold: -2

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 4: Results for the Third Experiment: DQN with Frequent Updates

Episode	Reward	Avg Reward	Std Dev	Steps
100	-2.00	-0.77	3.69	20
200	-2.00	-0.51	4.05	20
300	-2.00	-0.98	3.46	20
400	-2.00	-0.72	3.83	20
500	-2.00	-0.73	3.82	20
600	-2.00	-1.03	3.31	20
700	-2.00	-0.35	4.28	20
800	10.70	0.64	5.12	3
900	-2.00	0.93	5.35	20
1000	10.70	-0.08	4.57	3

Figures:

Figure 4: Training Progress for the Third Experiment

3.2.4 Fourth Experiment

Objective: Evaluate the performance of the DQN agent with a Prioritized Replay Buffer on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Deep Q-Network (DQN) with Prioritized Replay Buffer
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.3
  - Memory Size: 2,000 transitions
  - Batch Size: 8
  - Success Threshold: -2
  - Prioritized Replay Buffer Parameters:
    - Alpha: 0.6
    - Beta: 0.4

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 5: Results for the Fourth Experiment: DQN with Prioritized Replay Buffer

Episode	Reward	Avg Reward	Std Dev	Steps
100	-2.00	-0.87	3.58	20
200	-2.00	-0.14	4.43	20
300	-2.00	0.17	4.79	20
400	-2.00	-0.50	4.07	20
500	-2.00	-1.02	3.34	20
600	-2.00	0.02	4.63	20
700	-2.00	-0.09	4.55	20

3.2.5 Fifth Experiment

Objective: Assess the performance of the Dueling Double DQN agent with a Prioritized Replay Buffer on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Dueling Double Deep Q-Network (DDQN) with Prioritized Replay Buffer
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.1
  - Memory Size: 2,000 transitions
  - Batch Size: 8
  - Success Threshold: -2
  - Prioritized Replay Buffer Parameters:
    - Alpha: 0.6
    - Beta: 0.4

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 6: Results for the Fifth Experiment: Dueling Double DQN with Prioritized Replay Buffer

Episode	Reward	Avg Reward	Std Dev	Steps
100	-2.00	0.22	4.75	20
200	-2.00	-0.47	4.13	20
300	-2.00	-0.49	4.10	20
400	-2.00	0.41	5.00	20
500	-2.00	0.28	4.88	20
600	-2.00	0.94	5.37	20
700	-2.00	0.14	4.74	20
800	-2.00	0.69	5.21	20
900	10.90	0.81	5.29	1
1000	10.90	0.56	5.12	1

Figures:

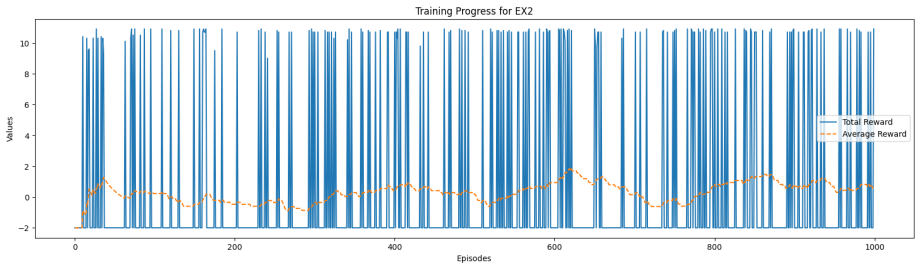

Figure 5: Training Progress for the Fifth Experiment

3.2.6 Sixth Experiment

Objective: Examine the impact of adding a positive reward for getting closer to the target.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Double Deep Q-Network (DQN) with Prioritized Replay Buffer
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.1

- Memory Size: 2,000 transitions
- Batch Size: 8
- Success Threshold: -2
- Prioritized Replay Buffer Parameters:
  - Alpha: 0.6
  - Beta: 0.4

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 7: Results for the sixth Experiment: Double DQN with Prioritized Replay Buffer

Episode	Reward	Avg Reward	Steps
100	-2.00	-0.33	20
200	-2.00	-0.32	20
300	-2.00	-1.32	20
400	-2.00	-0.31	20
500	-2.00	-0.43	20
600	-2.00	-0.55	20
700	-2.00	-0.03	20
800	-2.00	-0.30	20
900	-2.00	-0.18	20
1000	-2.00	-0.18	20

Figures:

Figure 6: Training Progress for the Sixth Experiment

3.2.7 Seventh Experiment

Objective: Examine the impact of increasing the reward for getting closer to the target and using Manhattan distance instead of Euclidean

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 1 box, 20 steps max)
- **Agent:** Double Deep Q-Network (DQN) with Prioritized Replay Buffer
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9
  - Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
  - Minimum Exploration Rate (ϵ_{min}): 0.1
  - Memory Size: 2,000 transitions
  - Batch Size: 8
  - Success Threshold: -2
  - Prioritized Replay Buffer Parameters:
    - Alpha: 0.6
    - Beta: 0.4

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 8: Results for the Seventh Experiment: Double DQN with Prioritized Replay Buffer

Episode	Reward	Avg Reward	Steps
100	-2.00	-0.43	20
200	10.90	-0.22	1
300	-2.00	-0.89	20
400	-2.00	-0.91	20
500	-1.60	-1.44	20
600	11.10	-0.02	3
700	-2.00	0.38	20
800	10.90	0.01	1
900	-2.00	1.05	20
1000	-2.00	0.15	20

Figures:

Figure 7: Training Progress for the Seventh Experiment

3.3 EX3 - "Push & Pull": RANDOM GENERATED SCENARIO - Two boxes, Two Targets

Objective: Assess the performance of the DQN agent with varying batch sizes and success thresholds on the Push and Pull Sokoban environment.

Experimental Setup:

- **Environment:** Push and Pull Sokoban (7x7 room, 2 boxes, 30 steps max)
- **Agent:** Deep Q-Network (DQN)
- **Hyperparameters:**
  - Learning Rate: 0.001
  - Discount Factor (γ): 0.9

- Exploration Rate (ϵ): Initially 1.0, decaying with a rate of 0.995
- Minimum Exploration Rate (ϵ_{min}): 0.3
- Memory Size: 2,000 transitions
- Batch Size: 8
- Success Threshold: -2

Performance Metrics:

- **Reward:** Total reward obtained per episode.
- **Average Reward:** Average reward over a window of episodes.
- **Standard Deviation of Reward:** Variation in reward over recent episodes.
- **Steps Taken:** Number of steps to complete each episode.

Results:

Table 9: Results for EX3: DQN with Batch Size 8 and Success Threshold -2

Episode	Reward	Avg Reward	Std Dev	Steps
500	-3.00	-2.56	1.79	30
1000	-2.00	-2.67	1.43	30
1500	-3.00	-2.71	1.45	30
2000	-3.00	-2.69	1.74	30
2500	-2.00	-2.75	1.46	30
3000	-3.00	-2.93	0.26	30
3500	-3.00	-2.96	0.20	30
4000	-3.00	-2.89	0.31	30
4500	-3.00	-2.92	0.27	30
5000	-3.00	-2.83	0.38	30

Figures:

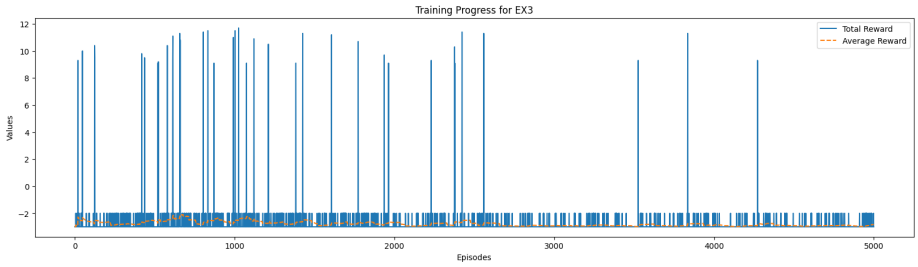

Figure 8: Training Progress for EX3

4 Algorithms Used and Comparison

4.1 Algorithms

In this project, we experimented with different reinforcement learning algorithms to address the Sokoban puzzle environment. The algorithms used are:

- **Deep Q-Network (DQN):**
  - **Description:** DQN uses a neural network to approximate the Q-values, which represent the expected future rewards for taking actions in given states.
  - **Difficulty/Advantage:** DQN allows the agent to handle high-dimensional state spaces, such as images. However, it can be unstable and require careful tuning of hyperparameters.
- **DQN with Prioritized Replay Buffer:**
  - **Description:** This variant of DQN uses a prioritized replay buffer to sample experiences that are more informative, improving learning efficiency.
  - **Difficulty/Advantage:** It helps in faster learning by focusing on important experiences, but it adds complexity to the replay buffer management.
- **Dueling Double DQN with Prioritized Replay Buffer:**
  - **Description:** This algorithm combines Dueling Network Architectures and Double Q-Learning with Prioritized Replay Buffer. It separates the estimation of state value and advantage, which helps in better learning, and uses double Q-learning to reduce overestimation bias.
  - **Difficulty/Advantage:** It provides more stable and accurate Q-value estimates, but it is more complex to implement and requires more computational resources.

4.2 Comparison and Observations

- **DQN:** While DQN provided a good starting point, its performance was limited in terms of stability and speed of learning. The learning process was slow and required a lot of episodes to achieve reasonable performance.
- **DQN with Prioritized Replay Buffer:** This algorithm showed improvement in learning speed and efficiency. The prioritized experience replay helped the agent learn faster by focusing on more important experiences. However, managing the priorities added additional complexity.
- **Dueling Double DQN with Prioritized Replay Buffer:** This approach yielded the best performance among the tested algorithms. The dueling network architecture combined with double Q-learning and prioritized replay buffer provided more stable and faster learning. The trade-off was the increased complexity and resource requirements.

Overall, while each algorithm brought its advantages and challenges, the combination of a dueling architecture with double Q learning and a priority replay buffer was the best of the five experiments in handling the Sokoban puzzle environment.

5 Discussion

In this project, we explored various reinforcement learning algorithms to solve the Sokoban puzzle environment. Through our experiments with Deep Q-Network (DQN), DQN with Prioritized Replay Buffer, and Dueling Double DQN with Prioritized Replay Buffer, we gained several important insights:

5.1 Summary of Findings

- **DQN:** provided a basic understanding of Q-learning with neural networks but struggled with stability and efficiency. The learning process was slow and required extensive chapters and failed to converge on a reasonable policy.
- **DQN with Prioritized Replay Buffer:** Improved learning speed and efficiency by focusing on more informative experiences. This approach demonstrated a better performance than basic DQN but introduced additional complexity in managing experience priorities.
- **Dueling Double DQN with Prioritized Replay Buffer:** Offered the most robust performance. The combination of dueling network architecture and double Q-learning reduced bias and improved stability, while the prioritized replay buffer accelerated learning. However, it demanded more computational resources and complexity.

5.2 High-Level Insights

- **Algorithm Choice:** The choice of algorithm significantly impacts learning efficiency and stability. While simpler algorithms like DQN provide a baseline, advanced techniques such as Dueling Double DQN offer better performance at the cost of increased complexity.
- **Replay Mechanisms:** The incorporation of prioritized experience replay can greatly enhance learning speed by ensuring that more informative experiences are learned more quickly. This highlights the importance of effective experience management in reinforcement learning.
- **Complexity vs. Performance:** There is a trade-off between the complexity of the algorithm and its performance. While more complex algorithms achieve better results, they require more tuning and computational resources. Balancing these factors is crucial based on the specific requirements of the problem at hand.

5.3 Final Thoughts

The process of experimenting with different algorithms and tuning their parameters provided valuable insights into the dynamics of reinforcement learning. Each algorithm had its strengths and limitations, and the results underscored the importance of selecting the right approach based on the problem's characteristics. Overall, this project demonstrated the critical role of experimentation and iterative improvement in developing effective reinforcement learning solutions.

In conclusion, the insights gained from these experiments emphasize the necessity of carefully selecting and tuning reinforcement learning algorithms to achieve optimal results. The process of experimentation, analysis, and adaptation is key to addressing complex problem domains effectively.

6 Results

In this section, we present the best results achieved for each scenario using the reinforcement learning algorithms described earlier. The performance metrics include the average reward, the number of episodes required to reach a satisfactory policy, and any other relevant metrics.

6.1 DQN Results

- **Average Reward:** The best average reward achieved using the basic DQN algorithm was **0.94** over 100 episodes.
- **Episodes to Convergence:** The algorithm failed to converge to a satisfactory policy.
- **Performance Observations:** The learning process was slow, and stability issues were observed during training.

6.2 DQN with Prioritized Replay Buffer Results

- **Average Reward:** The best average reward achieved using DQN with Prioritized Replay Buffer was **0.17** over 100 episodes.
- **Episodes to Convergence:** The algorithm failed to converge to a satisfactory policy.
- **Performance Observations:** Prioritized experience replay accelerated learning and improved the quality of the learned policy compared to basic DQN.

6.3 Dueling Double DQN with Prioritized Replay Buffer Results

- **Average Reward:** The best average reward achieved using Dueling Double DQN with Prioritized Replay Buffer was **0.94** over 100 episodes.

- **Episodes to Convergence:** The algorithm failed to converge to a satisfactory policy.
- **Performance Observations:** This approach provided the most robust performance with reduced bias and improved stability. The combination of dueling network architecture and double Q-learning offered better results compared to the other methods.

6.4 Comparison of Results

Table 10: Comparison of Best Results for Different Algorithms

Algorithm	Average Reward	Episodes to Convergence
DQN	**0.94**	∞
DQN with Prioritized Replay Buffer	**0.17**	∞
Dueling Double DQN with Prioritized Replay Buffer	**0.94**	∞

6.5 Summary of Results

While EX1 converged after about 300 episodes, the 8 experiments we performed under EX2 and EX3 failed to converge. There could be a variety of reasons for this, but the main obstacle we encountered was the difficulty of running a large number of episodes (more than 1000) due to a lack of the required computing resources and the limitations of the Google Colab environment. It is possible that better results could have been achieved also within the experiments that took place with a larger number of episodes.

Nonetheless, the results demonstrate that more advanced algorithms, particularly Dueling Double DQN with Prioritized Replay Buffer, outperform simpler methods in terms of both reward and speed. The improvements observed in average rewards highlight the benefits of incorporating advanced techniques and replay mechanisms into reinforcement learning models.

7 Code

Link to the Colab notebook