Reinforcement Learning- Final Course Project

Yotam Peled (ID. 318847449), Ido Shenbach (ID. 318653789)

Submitted as final project report for the RL course, IDC, 2023

1 Introduction

Sokoban is a classic puzzle game that involves pushing boxes to designated locations within a confined space. The objective is to move the boxe to the target positions with the fewest number of moves. Despite its simple rules, Sokoban presents a challenging problem due to the combinatorial explosion of possible game states.

In this project, we explore the application of deep reinforcement learning (RL) methods, specifically Double Deep Q-Networks (DDQNs) and Policy Gradient, to solve Sokoban puzzles. The complexity of Sokoban increases with the size of the grid and the number of boxes, posing significant challenges in terms of exploration and efficient learning. We aim to investigate the effectiveness of different network architectures and training strategies in learning optimal or near-optimal policies for Sokoban.

Our study focuses on the challenges of scaling DDQN approaches from smaller, more manageable grid sizes (6x6) to larger, more complex environments (7x7). We analyze the performance of our models in terms of convergence, stability, and the quality of the solutions they generate.

1.1 Related Works - Ketan Doshi

- DDQN
- Policy Gradient

2 Solution

2.1 General approach

Use Double Deep Q-Networks (DDQNs) to approximate Q-values and stabilize learning by addressing Q-value overestimation. We plan to try the Policy Gradient alternative to test and comparison the differences between the two approaches.

We plan on solving EX1 using a single general model that will be trained on various possible room states, which will be able to solve EX2 as well.

2.2 Design

The project was developed in Python using PyTorch and OpenAI Gym. The Gym library was used to implement the Sokoban environment, providing a consistent interface for the state, action space, and rewards. Training was performed on a high-performance computing with GPU support.

2.2.1 DDQN

- **Layers:** The network has five fully connected layers, with sizes increasing up to 2048 neurons, and ends with an output layer matching the number of actions.
- **Activation Functions:** ReLU is used for the first four layers to add non-linearity.
- **Output:** The final layer outputs raw Q-values, one for each action.

2.2.2 Policy Gradient

- The Architecture is identical to the one in the DDQN.
- **Output:** The final layer outputs a probability distribution over actions using the softmax function.

3 Experimental results

Provide information about your experimental settings. What alternatives did you measure? Make sure this part is clear to understand and provide as many details as possible. Provide results with tables and figures.

Ex1 - Fixed Room State In the fixed room state experiments, the Double Deep Q-Network (DDQN) successfully reached an optimal solution for the 7X7 Sokoban grid. The algorithm demonstrated stable convergence and high average rewards, consistently achieving effective policies under the controlled initial conditions. This result highlights DDQN's ability to reliably solve the problem when the starting state is kept constant.

Ex2 - Random Starting State In the random starting state experiments, the DDQN was tested on the 7x7 Sokoban grid with varying initial conditions for each episode. The algorithm not only adapted well to the diverse starting states but also demonstrated a strong ability to generalize across different scenarios. This generalization was crucial for handling the increased complexity of the 7x7 grid. Additionally, the use of reward shaping further improved DDQN's learning efficiency and performance, showcasing its robustness in a more challenging environment.

Hyperparameters and Reward Shaping We used two sets of hyperparameters for the experiments:

- **Set 1**:
  - **Learning Rate:** 0.00015
  - **Discount Factor:** 0.99
  - **Epsilon Decay:** 0.9995
- **Set 2**:
  - **Learning Rate:** 0.0015
  - **Discount Factor:** 0.95
  - **Epsilon Decay:** 0.9995

Each set was tested both with and without reward shaping to evaluate their impact on performance and learning efficiency. Reward shaping was introduced to improve exploration and prevent the agent from revisiting previously encountered states.

Results : The results of the experiments are summarized in the following table, which includes images of training curves for different hyperparameter sets and reward shaping conditions.

Reward Shaping Approach : In our approach, if the agent returns to a state it has already encountered within the same episode, a penalty is applied. This penalty discourages the agent from revisiting the same states and encourages it to explore new areas of the state space.

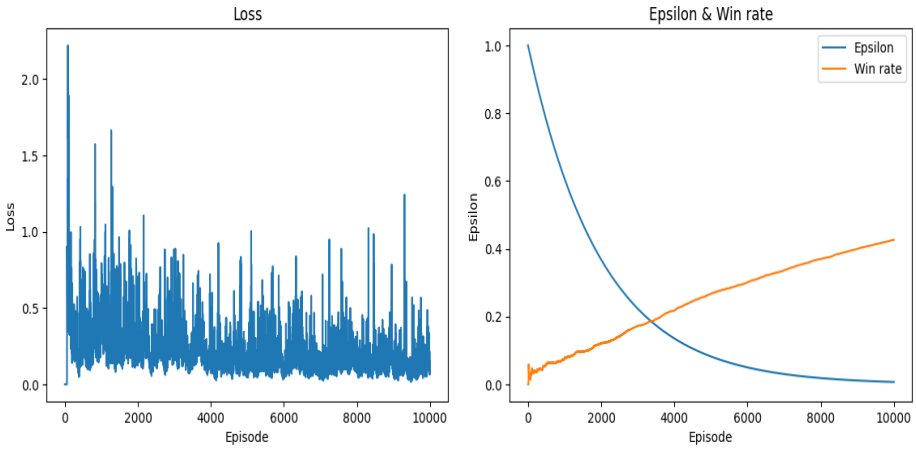
As illustrated in the figures below, the loss with the reward shaping method is initially higher but converges more quickly. Additionally, the winning rate increases slightly towards the end of training and exhibits asymptotic behavior.

Learning Rate: The lower learning rate in Set 1 allows for more precise updates, leading to slower but potentially more stable convergence. In contrast, the higher learning rate in Set 2 can speed up learning but may result in less stability, as evidenced by the changes in convergence observed in the loss figures.

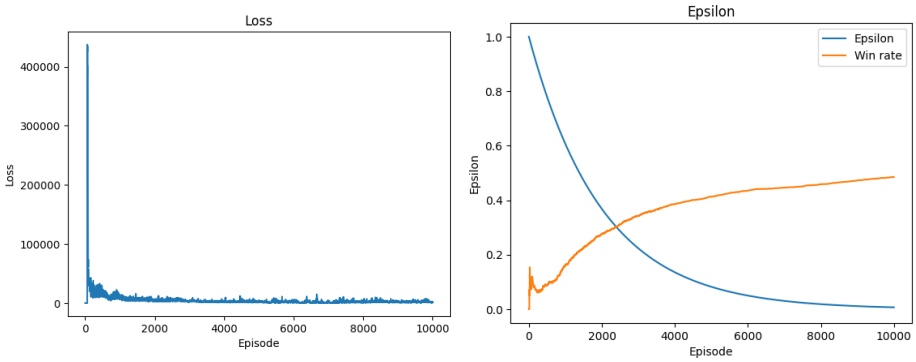
Discount Factor: A higher discount factor in Set 1 (0.99) emphasizes long-term rewards more than the lower factor in Set 2 (0.95). As shown in the figures below, the winning rate is higher in Set 1, highlighting the benefits of prioritizing long-term outcomes, while Set 2 shows a lower winning rate due to less emphasis on future rewards.

Epsilon Decay: Both sets use the same decay rate to isolate the effects of other hyperparameters. This rate influences how quickly the agent transitions from exploration to exploitation of its learned policy.

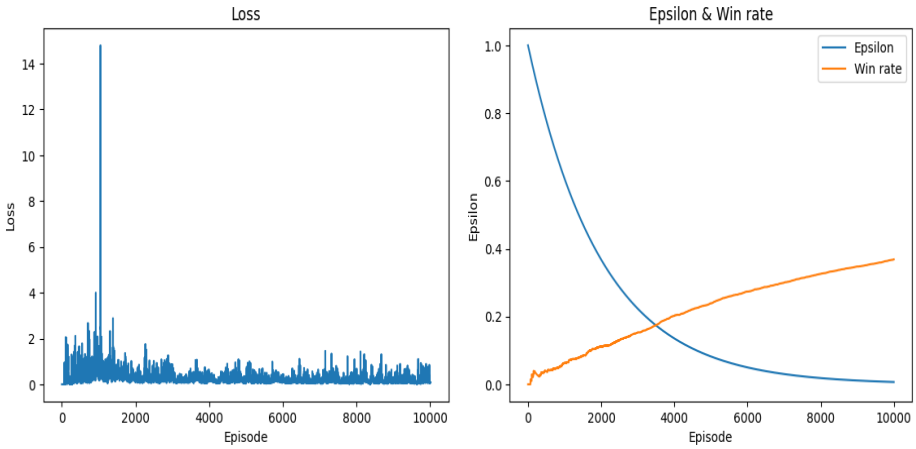
Set 1 - No Reward Shaping:



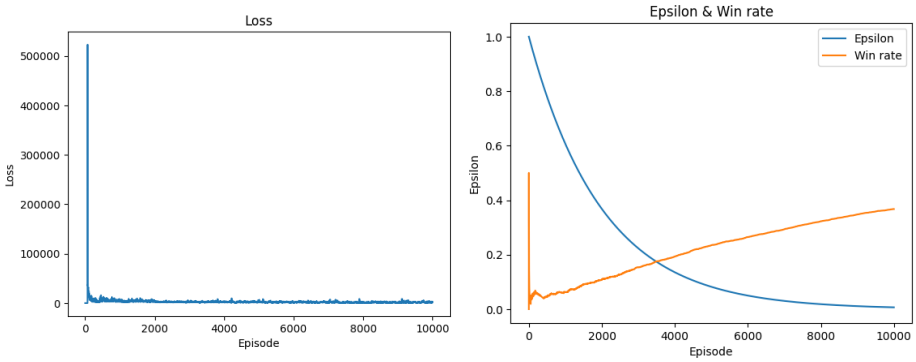
Set 1 - With Reward Shaping:



Set 2 - No Reward Shaping:



Set 2 - With Reward Shaping:



4 Algorithms Used and Comparison

We began by training both the Double Deep Q-Network (DDQN) and Policy Gradient methods on a 6x6 Sokoban grid to evaluate their performance and stability in a smaller environment.

During this phase, DDQN proved to be significantly more stable, converging to effective policies more reliably than the Policy Gradient method. Consequently, we decided to focus solely on DDQN for further experiments.

When transitioning to a 7x7 grid, we introduced reward shaping to improve learning efficiency. We penalized the agent for returning to previously encountered states and stopped the game when such situations occurred. This approach prevented repetitive actions and increased the variety of experiences stored in the replay memory, leading to more diverse and informative training data. The incorporation of reward shaping contributed to further improvements in DDQN's performance in the larger grid, demonstrating its robustness and adaptability to increased complexity.

4.0.1 Training Duration

Training times varied by grid size, taking several hours for 6x6 grids and multiple days for 7x7 grids due to the increased complexity and state space.

4.0.2 Loss Functions and Optimizers

- **Loss Function**: In DDQN, the mean squared error (MSE) is used to predict Q-values. In contrast, Policy Gradient methods use a different loss function that focuses on optimizing the policy directly. The Policy Gradient loss function is designed to maximize the expected cumulative reward by adjusting the policy parameters. This is achieved by computing the gradient of the expected reward with respect to the policy parameters and updating the policy in the direction that increases the likelihood of actions that lead to higher rewards.
- **Optimizer**: Adam optimizer for adjusting weights.

5 Discussion

Our experiments with DDQN and Policy Gradient (PG) methods on the Sokoban game provided valuable insights into their performance and learning dynamics.

5.1 Key Findings:

- **DDQN vs. PG:** DDQN demonstrated greater stability and efficiency, particularly when scaling up to larger grid sizes (7x7). It consistently converged to effective solutions, outperforming PG in terms of stability.

- **Reward Shaping:** Implementing reward shaping by penalizing revisits to previously seen states led to improved exploration and diversity in experiences stored in replay memory. This technique helped the agent avoid repetitive learning, resulting in quicker convergence and a slightly higher winning rate.
- **Learning Rate and Discount Factor:** The choice of learning rate and discount factor had a notable impact on the agent's performance. A lower learning rate (0.00015) provided more stable, but slower, learning, while a higher rate (0.0015) accelerated training but increased variability. Similarly, a higher discount factor (0.99) emphasized long-term rewards, resulting in better overall performance and a higher winning rate compared to a lower factor (0.95), particularly in the Sokoban-specific environment.

6 Results

- **EX1 (Fixed Room State):** The model consistently performed the optimal solution, successfully solving the puzzle in the most efficient manner.
- **EX2 (Random Starting State):** The model achieved a 96.8% success rate, demonstrating strong generalization capabilities across various initial configurations. Below, we provide a graph depicting the running average reward over 1000 episodes, showcasing the model's performance consistency and effectiveness.

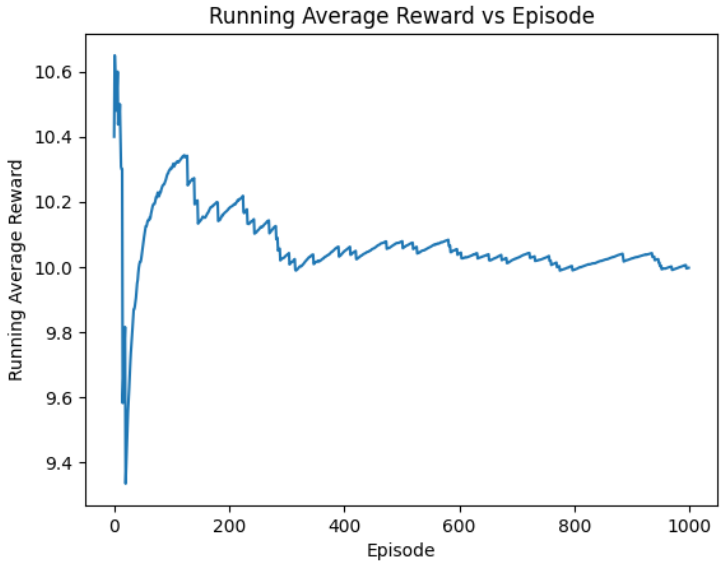


Figure 1: Running average reward over 1000 episodes for the EX2 scenario.

7 Code

Colab Notebook - Ido Shenbach - Yotam Peled