The islamic University of Gaza الجامعة الإسلامية بغزة

Faculty of IT كلية تكنولوجيا المعلومات

Sokoban AI agent with reinforcement learning

وكيل ذكي لحل لعبة سوكوبان باستخدام التعلم المعزز

By

محمد محمود محمد العبويني - 120210247

كريم عبد الكريم حسن محسن - 120210759

أحمد أشرف عبد الحي الخالدي - 120212269

Supervised by

أحمد محمد أحمد مرسى

A graduation project report submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology

October/2025

Abstract

This project focuses on the implementation of a reinforcement learning (RL) agent to solve the Sokoban puzzle environment. Sokoban is a grid-based puzzle that requires deep planning, where each wrong movement may make the level impossible to solve. The main goal of this project was to design and train an RL agent capable of solving Sokoban puzzles using multiple reinforcement learning algorithms such as PPO, RPPO, HRL, and DQN.

The project was completed over a period of eight weeks, starting from environment setup, and progressing through multiple experiments and models. Early experiments using PPO on complex environments (3 boxes and 10x10 grid). Later, simpler environments (1 box and 7x7 grid) were used to achieve better learning results. During training, several wrappers and reward-shaping techniques were applied to improve performance and avoid agent loops and deadlocks.

The results showed that the PPO model successfully solved the 1-box environment and achieved promising progress on 2-box environments. DQN demonstrated strong training improvement on both 1-box and 2-box environments but could not complete evaluation due to time constraints.

The main challenges faced during the project included limited hardware (4 GB GPU), slow training speed, and unstable internet and electricity conditions. Despite these limitations, the project achieved meaningful results and provided a strong foundation for future research on reinforcement learning for puzzle-solving environments.

ملخص الدراسة

يهدف هذا المشروع إلى تطوير وكيل ذكي باستخدام خوارزميات التعلم المعزز لحل لعبة "سوكوبان"، وهي لعبة تعتمد على التخطيط العميق والدقة في الحركة داخل بيئة شبكية. الهدف الرئيسي من المشروع هو تصميم وكيل تعلم معزز قادر على حل مستويات مختلفة من اللعبة باستخدام عدة خوارزميات مثل (PPO وDQN) وغيرها.

تم تنفيذ المشروع خلال فترة ثمانية أسابيع، بدأت بإعداد بيئة العمل وتنفيذ تجارب متعددة. فشلت المحاولات الأولى باستخدام خوارزمية PPO في بيئة معقدة (ثلاثة صناديق وشبكة 10 ,10)، وتم الانتقال لاحقاً إلى بيئات أبسط (مثل صندوق واحد وشبكة 7,7) للحصول على نتائج أفضل و وضع خط أساس لزيادة التعقيد لاحقاً.

تم تحسين أداء الوكيل من خلال تشكيل المكافآت (reward shaping) وإضافة قيود أو محددات (wrappers) لمعاقبة السلوكيات غير المرغوبة مثل الدوران المتكرر أو دفع الصندوق في الزوايا.

أظهرت النتائج النهائية أن النموذج PPO تمكن من حل بيئة الصندوق الواحد وحقق تقدماً ملحوظاً في بيئة الصندوقين، بينما أظهر النموذج DQN تحسناً مستمراً في التدريب في كلتا البيئتين، ولكن لم يتم تقييم النتائج النهائية بسبب ضيق الوقت.

رغم القيود المتعلقة بالوقت والمعدات، فقد وفر المشروع أساساً قوياً للأبحاث المستقبلية في مجال تطبيق التعلم المعزز على الألعاب والألغاز المعقدة.

Dedication

This project is dedicated to our families, friends, and teachers who supported and encouraged us during our academic journey.
We also dedicate this work to the Faculty of Information Technology at the Islamic University of Gaza for its continuous efforts in providing knowledge and guidance throughout our studies.
Their support and motivation inspired us to complete this project successfully.

Acknowledgment

We would like to express our deepest gratitude to our dear supervisor, *Mr. Ahmed Mohamed Ahmed Morsi*, for his guidance, patience, and valuable feedback throughout the duration of this project. His continuous support and technical advice helped us overcome many challenges.

We would also like to thank the Faculty of Information Technology at the Islamic University of Gaza for providing us with the opportunity and resources to complete this graduation project.

Finally, we extend our appreciation to our colleagues, friends, and everyone who contributed directly or indirectly to the completion of this work.

Table of Contents

المحتويات

List of Tables

List of Figures

List of Abbreviations

CNN	Convolutional Neural Network
DQN	Deep Q-Network
GPU	Graphics Processing Unit
HRL	Hierarchical Reinforcement Learning
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RPPO	Recurrent Proximal Policy Optimization
VRAM	Video Random Access Memory

Chapter 1
Introduction

Chapter 1
Introduction

Reinforcement learning (RL) is a subfield of artificial intelligence that focuses on how an agent can learn to make decisions by interacting with an environment and receiving rewards or penalties by made actions [1]. Sokoban is a popular and challenging environment used to test RL algorithms. It is a puzzle game where the agent must move boxes to specific target positions in a grid-based map without getting stuck.

The Sokoban environment is difficult for RL agents because it requires long-term planning and reasoning, where each wrong movement may block future solutions. In this project, the team explored how reinforcement learning can be applied to solve Sokoban under hardware and time constraints.

The project's aim was to train and evaluate multiple reinforcement learning models using different configurations to identify which algorithm performs best and how reward shaping affects performance.

1.1 Problem Statement

Training reinforcement learning agents for Sokoban is a difficult problem because the rewards are sparse and delayed. The agent often receives no feedback until the level is completely solved, which makes the training slow and unstable.
Another challenge is the large state space when the environment contains multiple boxes or larger grids, which increases the complexity and training time.

The project tackled these problems by applying different RL algorithms, simplifying the environment gradually, and using reward shaping and wrappers to penalize wrong behaviors such as looping or creating deadlocks.

1.2 Objectives

1.2.1 Main Objective

- To design and train a reinforcement learning agent capable of solving Sokoban puzzles using PPO, RPPO, HRL, and DQN algorithms [2], [3], [4].

1.2.2 Sub Objectives

- Configure and customize the Sokoban environment using the Gym and Gym-Sokoban libraries [5], [6].
- Implement multiple RL algorithms and compare their learning performance.
- Apply different reward shaping strategies to enhance agent learning.
- Test different environment configurations :
  (1-box, 2-boxes, 3-boxes : 7x7, 8x8, 10x10 grids).

- Analyze the learning curves and training behavior using Tensor Board.
- Evaluate the effect of algorithm type and environment complexity on the learning outcome.

1.3 Scope and Limitations

The project focused on training reinforcement learning models to solve Sokoban environments using Python and open-source RL libraries.
The study covered several models and configurations but was limited by available resources and time.

Main limitations included:

- 4 GB GPU (NVIDIA GTX 1650), which limited training duration and batch size.
- Internet and electricity instability during the project period.
- Long convergence time for reinforcement learning algorithms.
- Partial success in multi-box environments due to hardware and time constraints.

1.4 Importance of the project

This project provides valuable insights into how reinforcement learning can be applied to challenging environments that require logical reasoning and planning.
The results and analysis contribute to understanding the behavior of PPO and DQN under different Sokoban configurations.
It also shows how resource constraints can affect training performance and demonstrates methods to overcome such issues through reward shaping and code optimization.
This project can serve as a foundation for future work aiming to apply reinforcement learning in robotics, path planning, or puzzle-solving applications.

1.5 Tables

Table (1) : Scope and limitations

Area	In Scope	Constraints
Algorithms	DQN [2], PPO [3]	RPPO, HRL [4] limited by resources
Environments	1-box, 2-box	3-box not fully tested
Hardware	4GB GPU training	Limited model complexity
Time	8 weeks development	Insufficient for full evaluation

1.6 Figures

Figure (1): Reinforcement Learning Loop

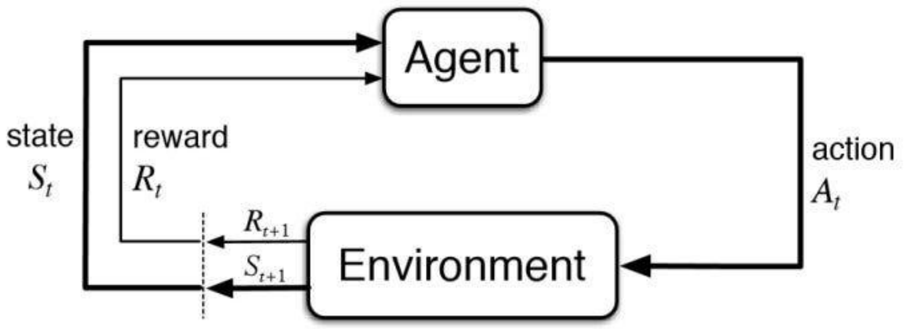


Figure (2): HRL with landmarks paper results on Boxoban training [4]

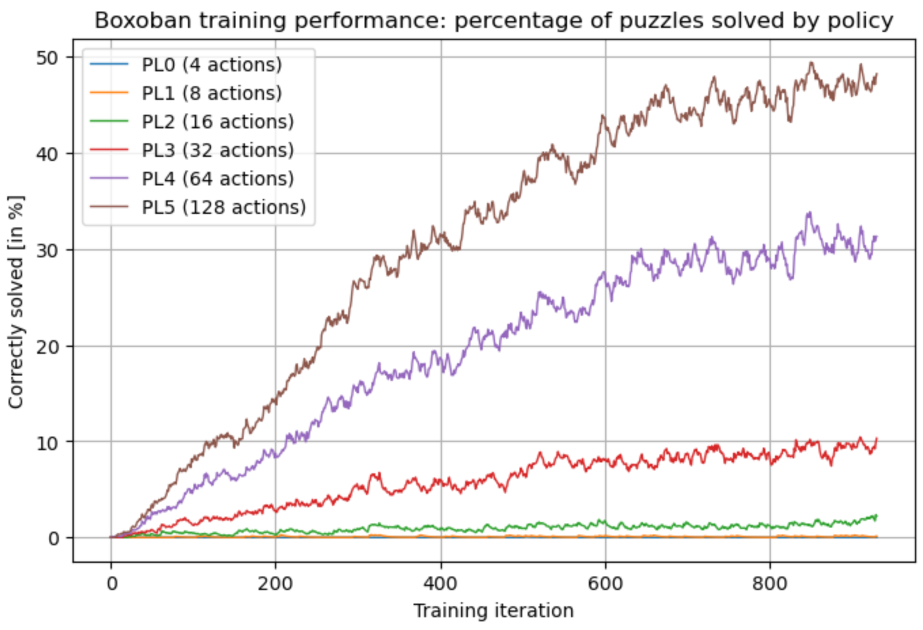


Figure (3): Successfully Solved Episode Using Gym-Sokoban [5]

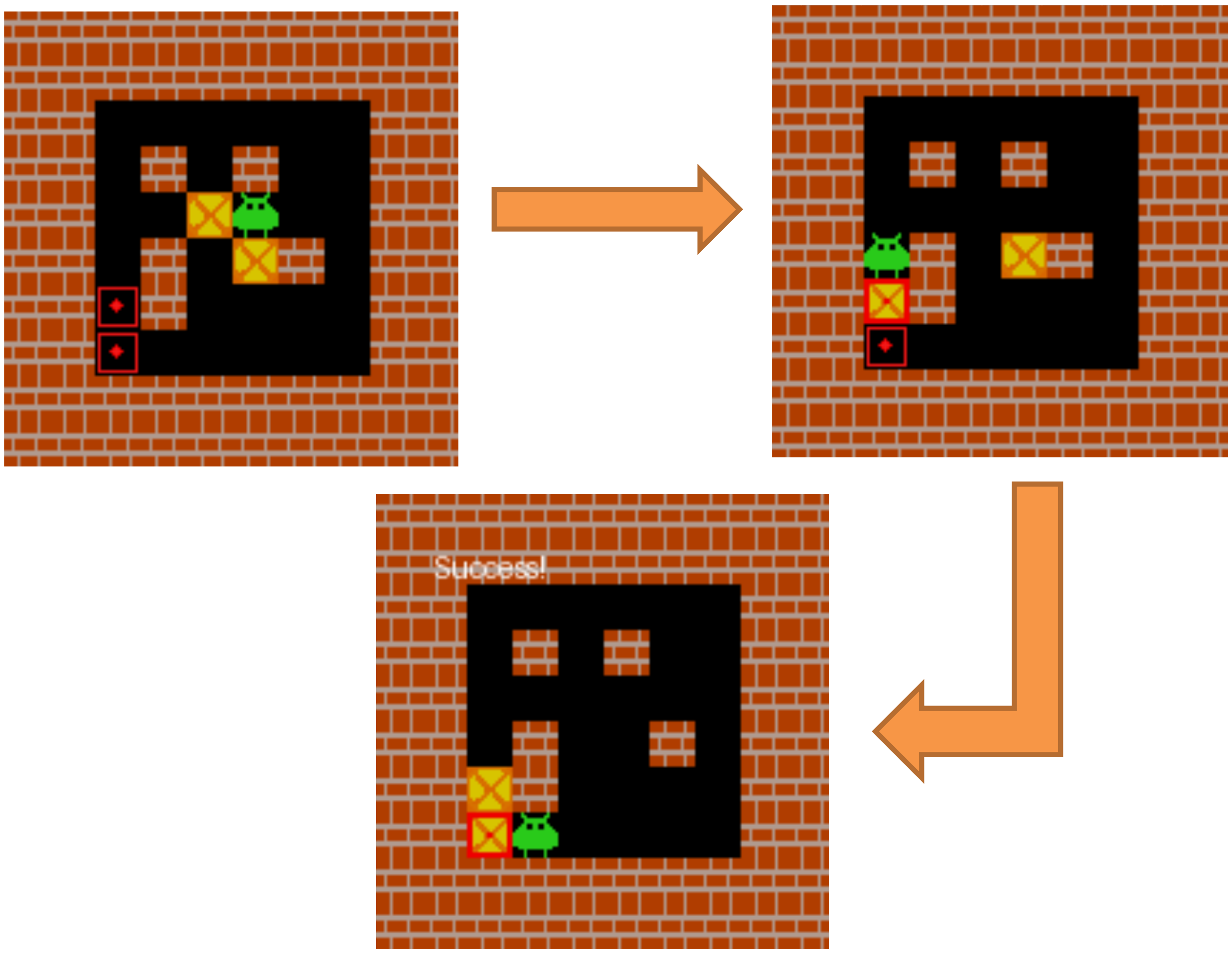
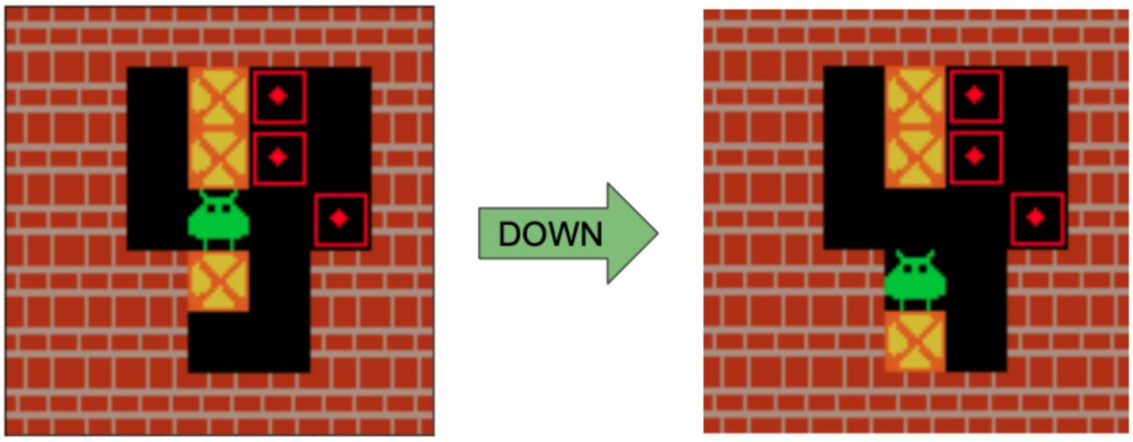


Figure (4): Single Wrong Move that Made a Level Unsolvable



Chapter 2
Related Works

Chapter 2

Related Works

Reinforcement learning (RL) has been successfully applied in many areas such as games, robotics, and control systems [1]. Classic examples include Atari games, chess, and the Go game, where RL agents achieved or surpassed human-level performance [2].
However, the Sokoban puzzle remains one of the hardest problems for RL because of its complex state space, delayed rewards, and the need for long-term reasoning.

Many previous studies have attempted to train reinforcement learning agents to solve Sokoban. Most used algorithms like PPO (Proximal Policy Optimization), A3C (Asynchronous Advantage Actor-Critic), or DQN (Deep Q-Network) [2], [3]. These algorithms perform well in simpler environments but struggle with Sokoban's high dimensionality and sparse feedback.

To address these challenges, some researchers applied reward shaping—giving the agent small intermediate rewards when it makes progress, such as moving a box closer to a goal or penalizing moves that push boxes into corners [1]. Others used curriculum learning, where the training starts from easy levels and gradually increases in difficulty, helping the agent to generalize better [1].

Despite these improvements, most existing implementations require very powerful hardware (e.g., GPUs with 16GB VRAM or more) and extremely long training times (often millions of timesteps) to reach consistent results.

In contrast, this project focuses on achieving useful progress under limited resources, using practical optimization techniques. The approach includes simplified environments, modified reward structures, and custom wrappers to detect looping or deadlock behaviors. We specifically evaluated PPO, DQN, RPPO, and HRL algorithms under constrained hardware conditions (4GB GPU VRAM), demonstrating that reinforcement learning can still succeed even under tight computational constraints [2], [3], [4].

Chapter 3
Methodology

Chapter 3
Methodology

3.1 Methodology Type

The project followed the Waterfall Model as its main software development methodology.

The Waterfall Model was chosen because it provides a clear and structured workflow, which suited the time-limited nature of the project. Each stage is completed before moving to the next, ensuring that the project progresses in an organized and traceable manner.

The stages of the Waterfall model applied in this project are:

1. Requirement Analysis
2. System Design
3. Implementation
4. Testing
5. Documentation

This sequence helped maintain focus, minimize confusion, and ensure every phase was properly completed before the next one began.

3.2 Tools and Equipment

The project was implemented using Python, supported by several specialized libraries for reinforcement learning, visualization, and analysis [6], [5], [7]. The details are in the tables section.

3.3 Team Management

The project was carried out collaboratively. Each team member contributed to multiple stages including setup, coding, and debugging. All members participated in training supervision, log analysis, and report preparation.
Team communication was maintained through shared sessions, regular updates, and problem-solving meetings.
This cooperative approach helped ensure project completion despite frequent challenges such as unstable internet and limited hardware.

3.4 Tables

Table (2): Timetable

Week	Main Activities
Week 1	Environment setup and installation of dependencies. Solved version conflicts between Gym and Gymnasium.
Week 2	First PPO experiments on 3-box (10×10) environment; training was not successful due to complexity.
Week 3	Shifted to 2-box setup, applied early reward shaping; no improvement achieved.
Week 4	Simplified to 1-box (7×7) environment, added wrappers for loops and deadlocks; achieved first working results.
Week 5	Moved to 2-box (8×8) setup; model showed progress but did not fully converge. Internet issues caused delays.
Week 6	Tried curriculum learning and RPPO; unsuccessful due to limited GPU (4 GB) and long runtime.
Week 7	Experimented with HRL; training started but results were unstable.
Week 8	Implemented and trained DQN; strong improvement observed, but evaluation was not completed due to time limits.

Table (3): Tools and Equipment

Tool / Library	Version	Purpose
Gym [6]	0.21.0	RL environment base
Gym-Sokoban [5]	0.0.7	Sokoban environment
Stable-Baselines3 [7], [8]	1.6.2	PPO and DQN algorithms
SB3-Contrib	1.6.2	RPPO and other extensions
Torch	latest	Deep learning framework
Torch Vision	latest	Model architecture support
NumPy	1.24.4	Numerical operations
TensorBoard [9]	latest	Monitoring and training visualization
Imageio[ffmpeg]	latest	Video recording for training runs
OpenCV-Python	latest	Frame handling and visualization

Chapter 4
Requirements Analysis

Chapter 4
Requirements Analysis

This chapter describes the requirements necessary to design, implement, and test the Sokoban reinforcement learning project.
It includes both functional requirements (what the system must do) and non-functional requirements (how the system should behave).
Clear requirement identification was essential to ensure that the system could be implemented efficiently within the limited time and hardware resources available.

4.1 Functional Requirements

Functional requirements define the specific operations and actions that the system should perform to meet its objectives.

Table (4): Functional Requirements

ID	Requirement	Description
FR1	Environment Setup	The system should initialize the Sokoban environment using Gym-Sokoban.
FR2	Agent Training	The system should train an RL agent using PPO, RPPO, HRL, and DQN algorithms.
FR3	Reward Shaping	The agent must receive shaped rewards to improve learning performance.
FR4	Loop and Deadlock Detection	The system should include wrappers to penalize undesired agent behaviors such as looping or pushing boxes into corners.
FR5	Logging and Visualization	The system should record training logs and display results using TensorBoard.
FR6	Model Saving and Loading	The system should save trained models and allow reloading for further training or evaluation.
FR7	Evaluation	The system should evaluate trained models in both training and test environments.

4.2 Non-Functional Requirements

Non-functional requirements describe the quality attributes and performance standards that the system should meet.

Table (5): Non-Functional Requirements

ID	Requirement	Description
NFR1	Usability	The system should be user-friendly and allow easy modification of parameters.
NFR2	Performance	The system should train efficiently despite limited hardware.
NFR3	Reliability	The training process should be stable and reproducible.
NFR4	Scalability	The system should support different environment sizes and numbers of boxes.
NFR5	Maintainability	The code should be modular and easy to update.
NFR6	Compatibility	The environment should be compatible with available Python and RL library versions.

4.3 System Constraints

The system was developed under real-world limitations related to hardware, time, and working conditions.
These constraints directly affected the scope and pace of the project.

Table (6): System Constraints

Constraint	Description
Hardware	GPU limited to 4 GB VRAM, reducing training batch sizes and runtime length.
Internet	Repeated disconnections during debugging and model downloads.
Electricity	Several interruptions affected long training sessions.
Time	Eight-week total duration limited experiment depth and model fine-tuning.

Chapter 5
System Design

Chapter 5

System Design

This chapter presents the overall system architecture and design of the Sokoban reinforcement learning project.
The system was structured in a modular way to ensure flexibility, scalability, and easy debugging.
Each module was designed to perform a specific function that contributes to the overall learning process of the agent.

5.1 System Architecture

The system architecture consists of several connected components that work together to train, evaluate, and visualize the performance of the reinforcement learning agent.
The modular design allowed experiments to be run with different algorithms, reward structures, and environment configurations without rewriting major parts of the code using Stable-Baselines3 framework [7], [8].

System Components

1. **Environment Module** – Responsible for initializing and managing the Gym-Sokoban environment.
2. **Agent Module** – Contains the reinforcement learning algorithm (PPO, RPPO, HRL, or DQN).
3. **Reward Shaping Module** – Adjusts the rewards and applies penalties for undesired actions.
4. **Wrapper Module** – Detects and penalizes looping and deadlock situations during training.
5. **Training Module** – Controls the training process, including hyperparameters, logging, and saving models.
6. **Evaluation Module** – Tests the performance of trained models under different configurations.
7. **Visualization Module** – Uses TensorBoard and OpenCV to visualize training progress, reward curves, and agent behavior.

5.2 System Workflow

The system workflow starts with environment initialization and model configuration. The agent interacts with the environment, receives observations, and performs actions.
Based on the result of each action, the agent receives a reward or penalty.
This process continues through multiple episodes until the agent learns optimal strategies to solve the environment.

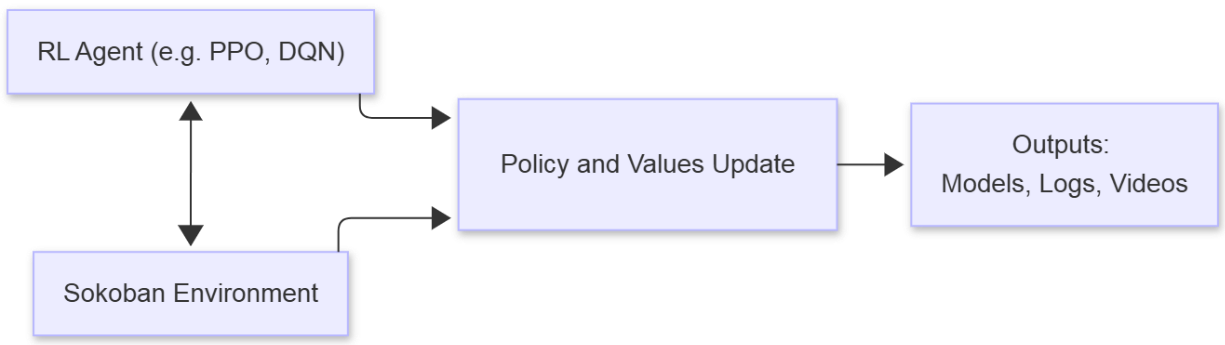
Workflow Steps

1. Initialize environment (Gym-Sokoban).
2. Select RL algorithm (PPO, RPPO, HRL, or DQN).
3. Configure parameters (learning rate, steps, reward structure).
4. Train the model through multiple episodes.
5. Log data to TensorBoard and save model checkpoints.
6. Evaluate trained model on test environments.
7. Visualize agent performance using OpenCV and recorded videos.

5.3 System Diagram

The following diagram illustrates the general design of the system, showing how each module interacts during training and evaluation.

Figure (5): Simplified System Diagram



5.4 PPO Agent Design

The PPO agent follows an actor-critic structure consisting of two neural networks [3]:

- **Actor Network:** Decides which action to take based on the current observation.
- **Critic Network:** Evaluates how good the action was by estimating the value function.

Training is performed by alternating between data collection (agent-environment interaction) and policy updates using clipped loss functions to maintain stable learning.

5.5 DQN Agent Design

The DQN agent uses a deep Q-network that estimates the Q-values for each possible action given a state [2].
It applies experience replay and target network updates to stabilize learning.
DQN is particularly effective for discrete action spaces like Sokoban, where the agent can move in a finite number of directions.

Chapter 6 Implementation and Testing

Chapter 6

Implementation and Testing

This chapter describes how the project was implemented and tested throughout the eight-week timeline.
It details the experiments, models, and methods used, along with key results, performance observations, and the challenges encountered during training.

6.1 Implementation Overview

The implementation phase began after successfully setting up the environment using Gym-Sokoban and Stable-Baselines3 [5], [7].
The system was developed using Stable-Baselines3, which provides reliable reinforcement learning implementations [7], [8].
All training sessions were executed on a personal computer with limited hardware (GTX 1650 GPU, 4 GB VRAM), which required optimizing training parameters for efficiency.

6.2 Environment Setup

During the first week, several installation and dependency conflicts occurred between Gym and Gymnasium [6].
After resolving these issues, the environment was successfully initialized.
Additional tools were integrated such as Imageio with ffmpeg for video recording and TensorBoard for visualizing training progress [9].

6.3 PPO Implementation and Results

At first, PPO was tested on a 3-box environment (10×10) with 300k timesteps, but the model couldn’t learn due to the high complexity and sparse rewards.
During the second and third weeks, the setup was simplified to 2 boxes, yet the results remained flat with no learning progress.

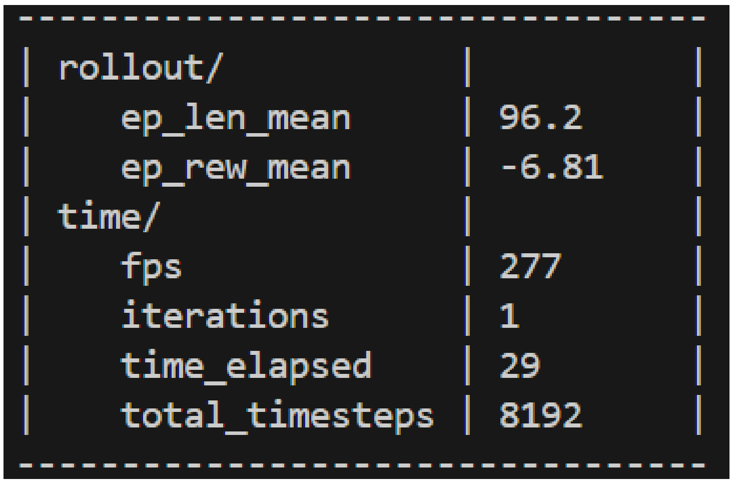
In the fourth week, the team reduced the problem to 1 box (7×7 grid) and simplified the reward function.

Custom wrappers were added to penalize unproductive behaviors such as:

- Looping**:** Repeated back-and-forth movement.
- Deadlocks**:** Pushing a box into a corner.
- Idle steps**:** Staying still without acting.

After applying these improvements, the PPO agent achieved over 90% success rate on 1-box environments, however it took way long time under limited resources and medium-end GPU.
However, training on 2 boxes (8×8) was only partially successful, it showed improvement but didn’t had enough time to improve fully due to limited runtime and hardware.

Figure (6): PPO Used Logging Pattern



```
---------------------------------
| rollout/           |          |
|    ep_len_mean     | 96.2     |
|    ep_rew_mean     | -6.81    |
| time/              |          |
|    fps             | 277      |
|    iterations      | 1        |
|    time_elapsed    | 29       |
|    total_timesteps | 8192     |
---------------------------------
```

6.4 RPPO and HRL Trials

In the sixth and seventh weeks, RPPO (Recurrent PPO) and HRL (Hierarchical Reinforcement Learning) were tested [4].
RPPO included LSTM layers to improve temporal understanding but consumed too much memory and time.
HRL required a more complex setup and didn't achieve clear learning progress within the limited available time.

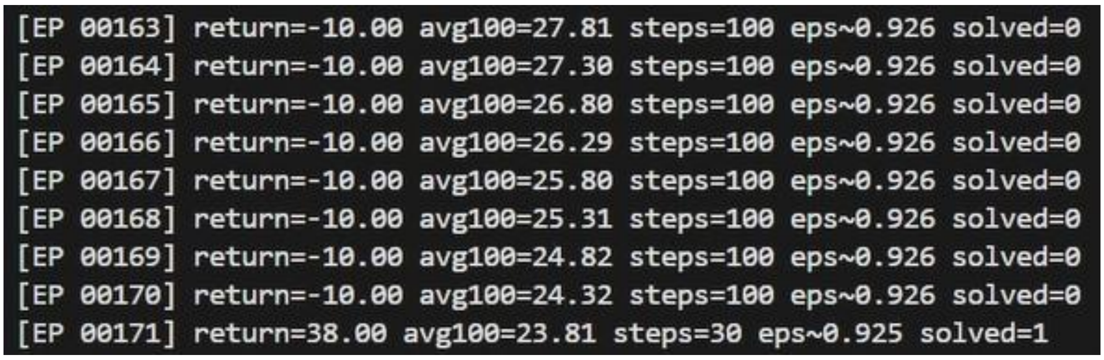
Both experiments provided useful insights but were discontinued due to resource constraints.

6.5 DQN Implementation and Results

In the eighth week, DQN was implemented and trained on both 1-box and 2-box environments.
During training, the model showed steady improvement and a clear increase in rewards, especially in the 1-box case.
The 2-box environment also displayed promising progress, but due to limited time, the evaluation phase could not be completed.

Most training logs indicated consistent learning and better stability compared to PPO.
It is expected that with longer training time and more computational power, DQN would achieve strong performance in evaluation as well.

Figure (7): DQN Used Logging Pattern



```
[EP 00163] return=-10.00 avg100=27.81 steps=100 eps~0.926 solved=0
[EP 00164] return=-10.00 avg100=27.30 steps=100 eps~0.926 solved=0
[EP 00165] return=-10.00 avg100=26.80 steps=100 eps~0.926 solved=0
[EP 00166] return=-10.00 avg100=26.29 steps=100 eps~0.926 solved=0
[EP 00167] return=-10.00 avg100=25.80 steps=100 eps~0.926 solved=0
[EP 00168] return=-10.00 avg100=25.31 steps=100 eps~0.926 solved=0
[EP 00169] return=-10.00 avg100=24.82 steps=100 eps~0.926 solved=0
[EP 00170] return=-10.00 avg100=24.32 steps=100 eps~0.926 solved=0
[EP 00171] return=38.00 avg100=23.81 steps=30 eps~0.925 solved=1
```

6.6 Reward Shaping and Wrappers

Reward shaping played a critical role in achieving stable learning.
By introducing penalties for repeated or stuck movements and bonuses for meaningful progress, the agent learned faster and avoided common mistakes.
This improvement was most noticeable when comparing early PPO runs (flat rewards) with later ones (clear upward trends).

6.7 Testing and Evaluation

Testing involved evaluating trained models on unseen Sokoban levels.
For 1-box environments, PPO achieved near-perfect solutions in most test cases.
For 2-box and higher environments, partial success was achieved, indicating the need for longer training durations.

Testing also included analyzing reward curves, loss stability, and video recordings to visually confirm agent behavior.

6.8 Training Charts

Figure (8): DQN 1-box average return

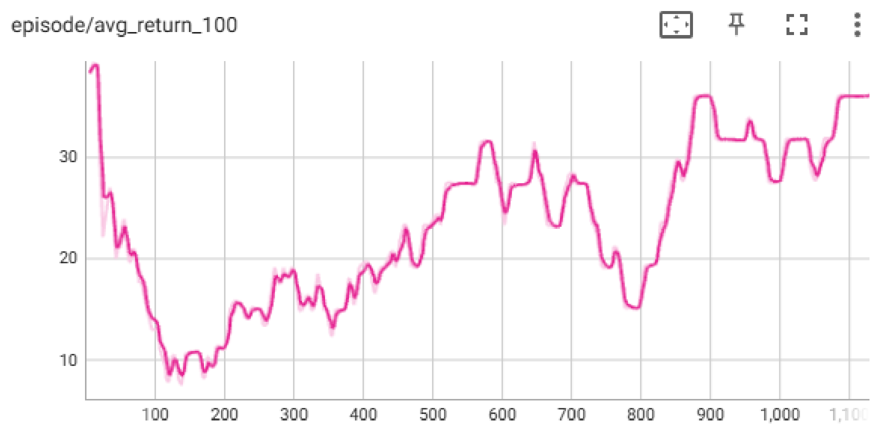


Figure (9): DQN 1-box epsilon (exploration rate)

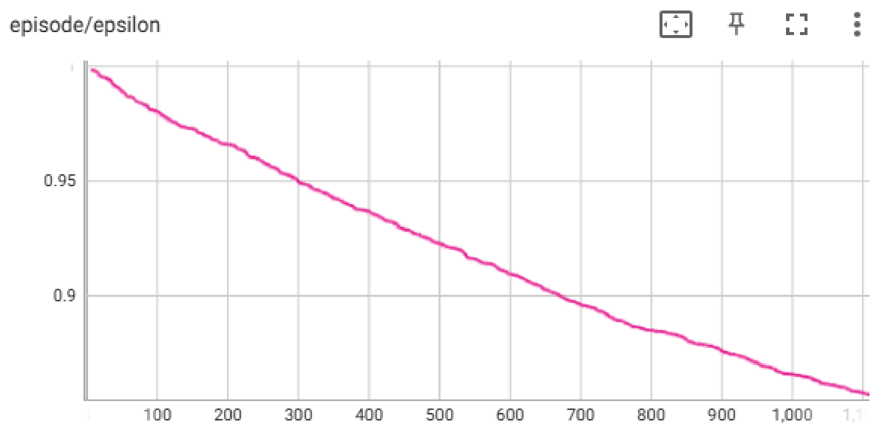


Figure (10): DQN 1-box loss chart

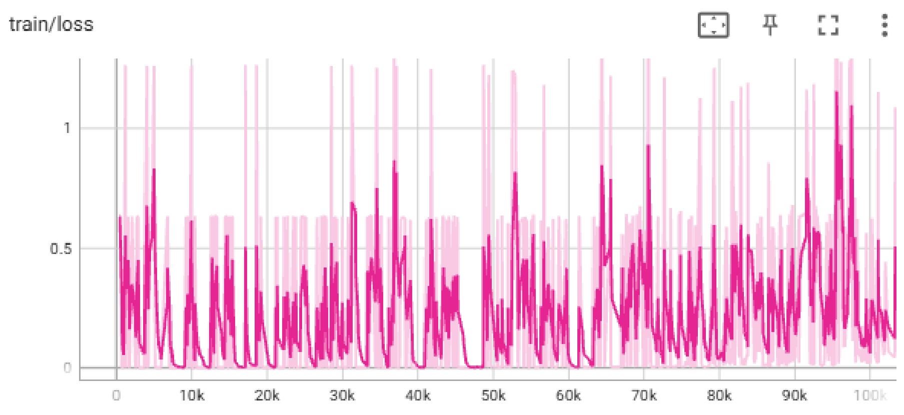


Figure (11): DQN 2-boxes average return

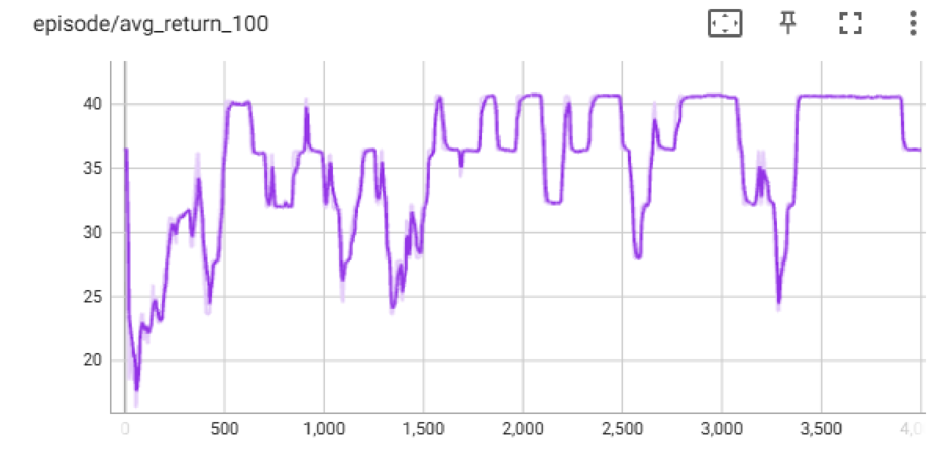


Figure (12): DQN 2-boxes return per episode

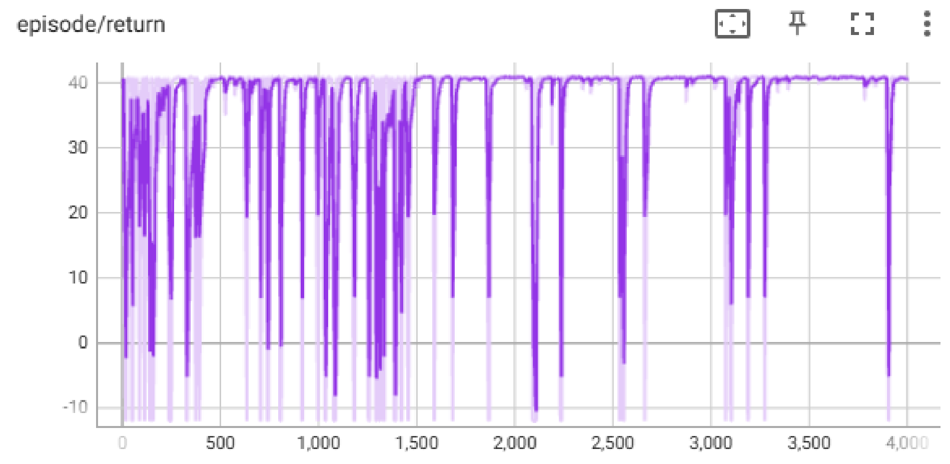


Figure (13): DQN 2-boxes length per episode

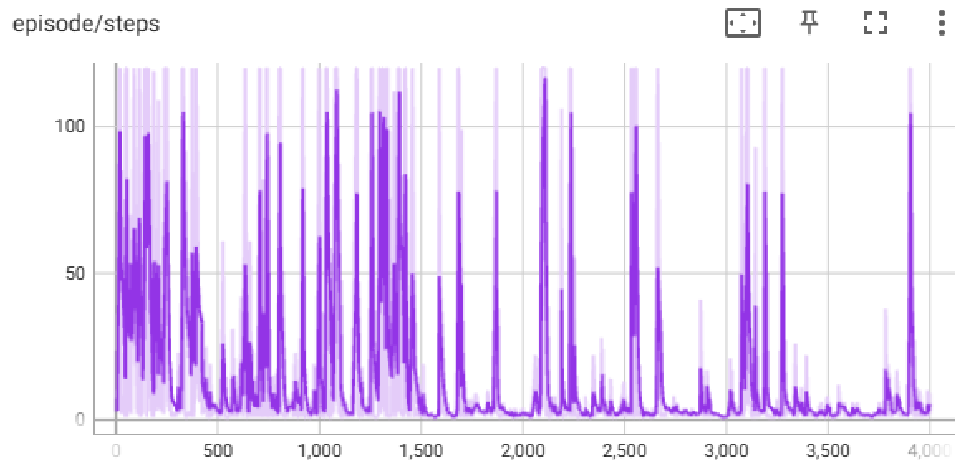


Figure (14): Curriculum episode reward mean

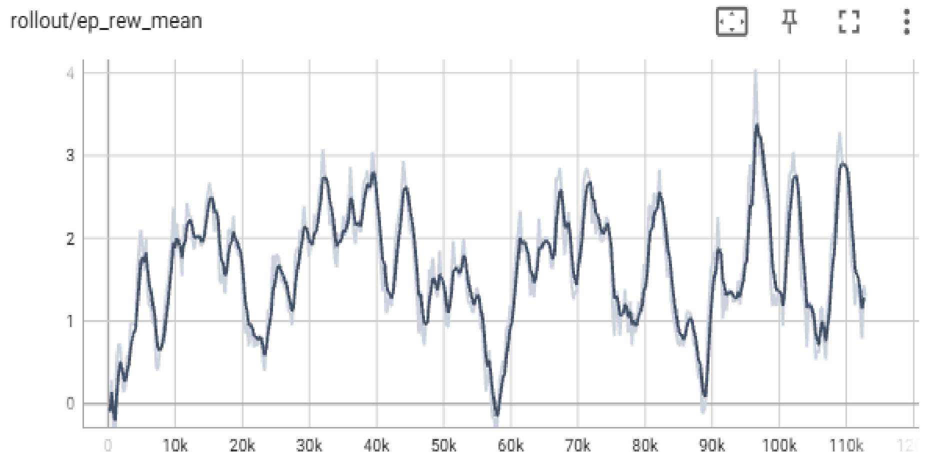


Figure (15): Curriculum episode length mean

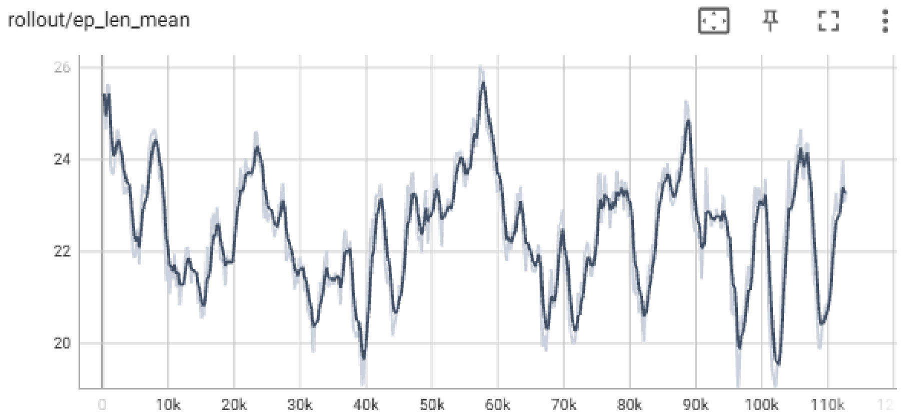


Figure (16): Curriculum entropy loss

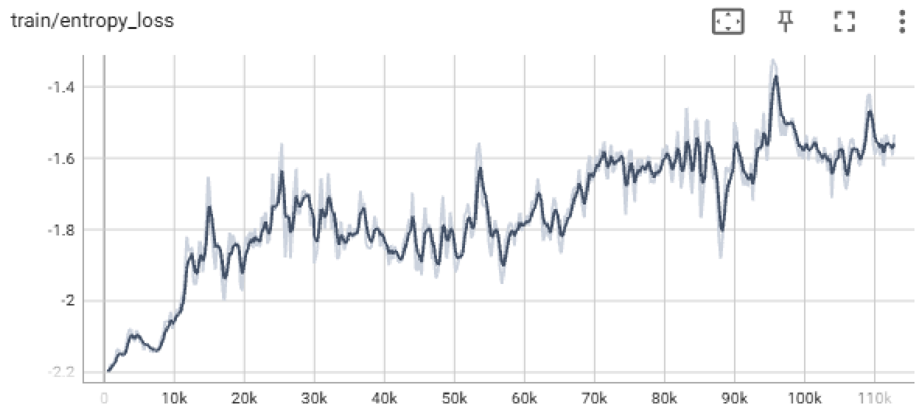


Figure (17): HRL reward per episode

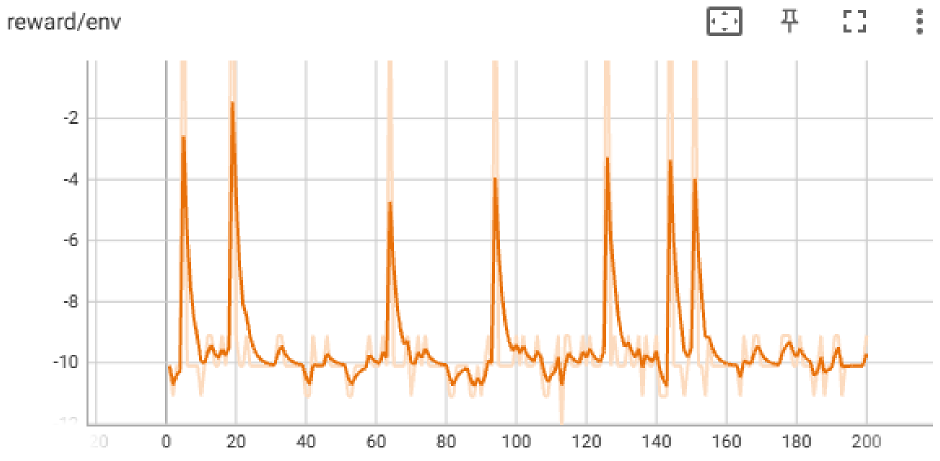


Figure (18): HRL entropy per episode

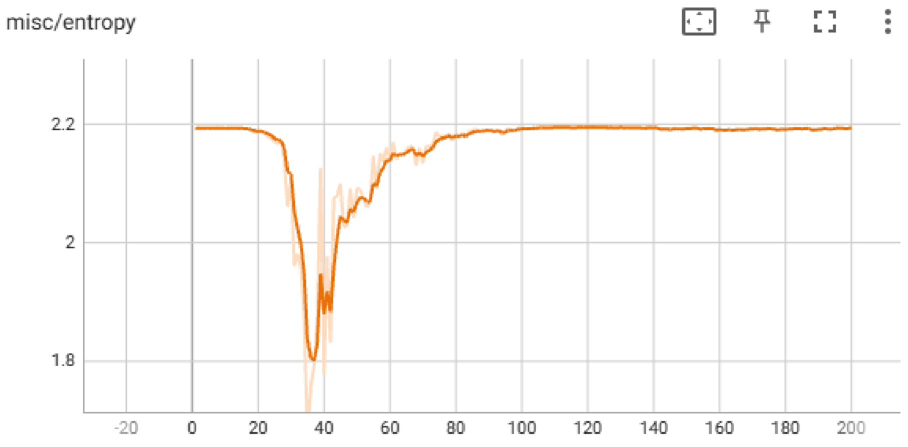


Figure (19): HRL episode length per episode

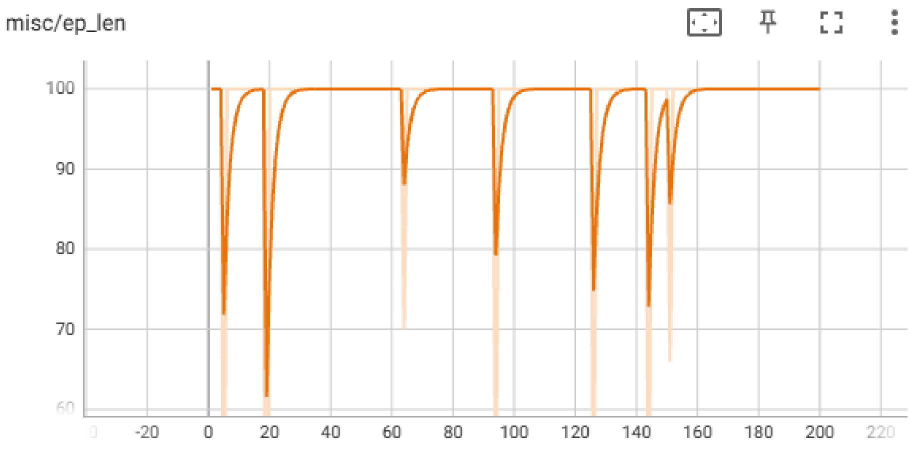


Figure (20): RPPO episode length mean

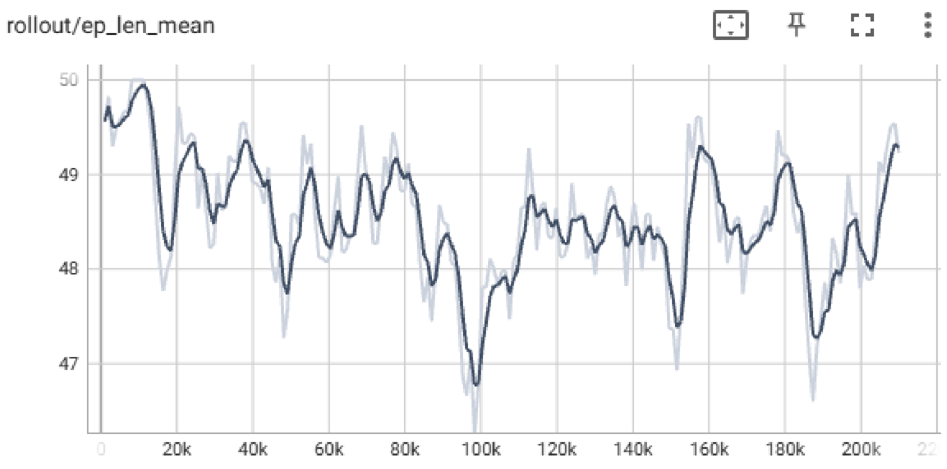


Figure (21): RPPO episode reward mean

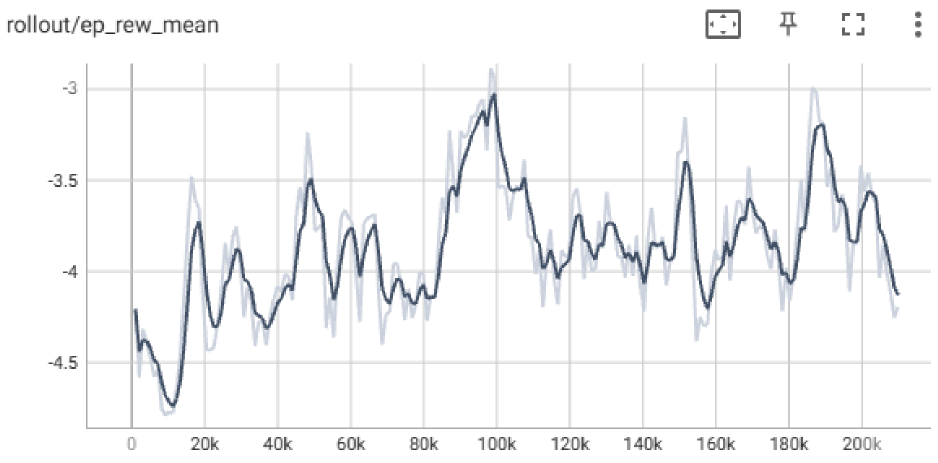


Figure (22): RPPO clip fraction

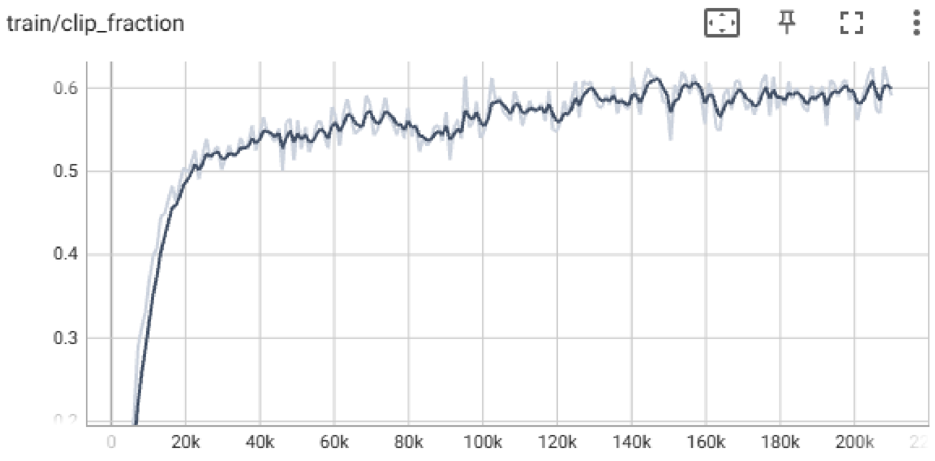


Figure (23): PPO 1-box episode length mean

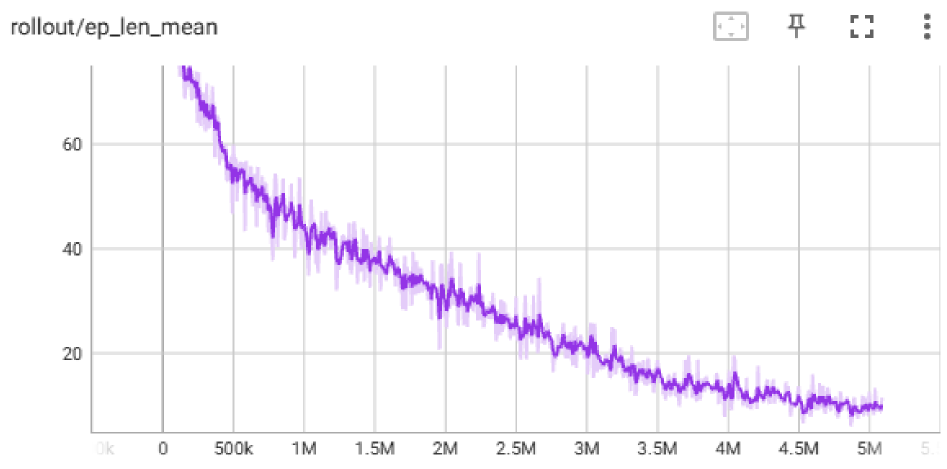


Figure (24): PPO 1-box episode reward mean

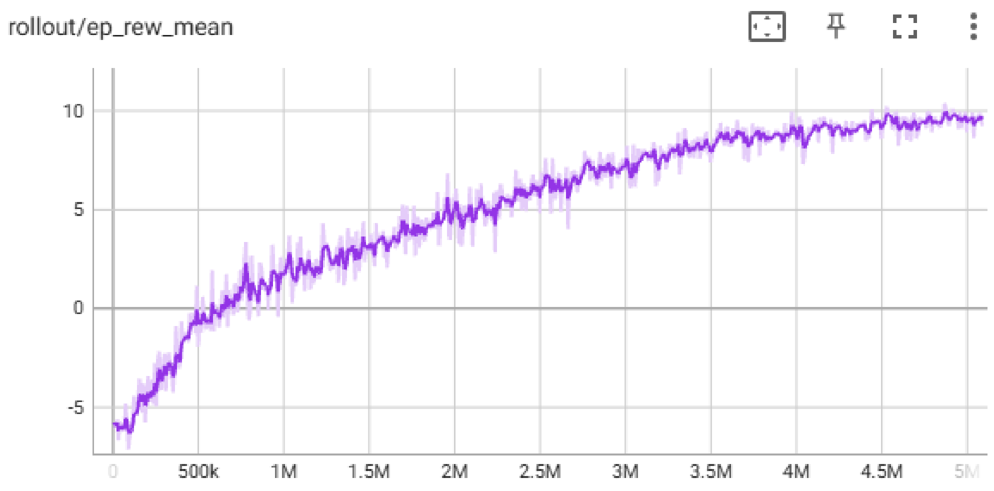


Figure (25): PPO 1-box clip fraction

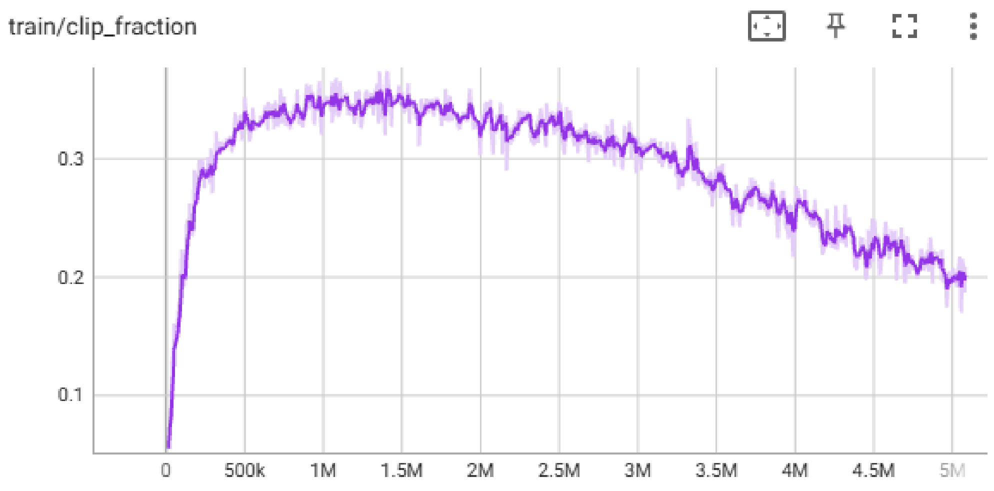


Figure (26): PPO 2-boxes episode length mean

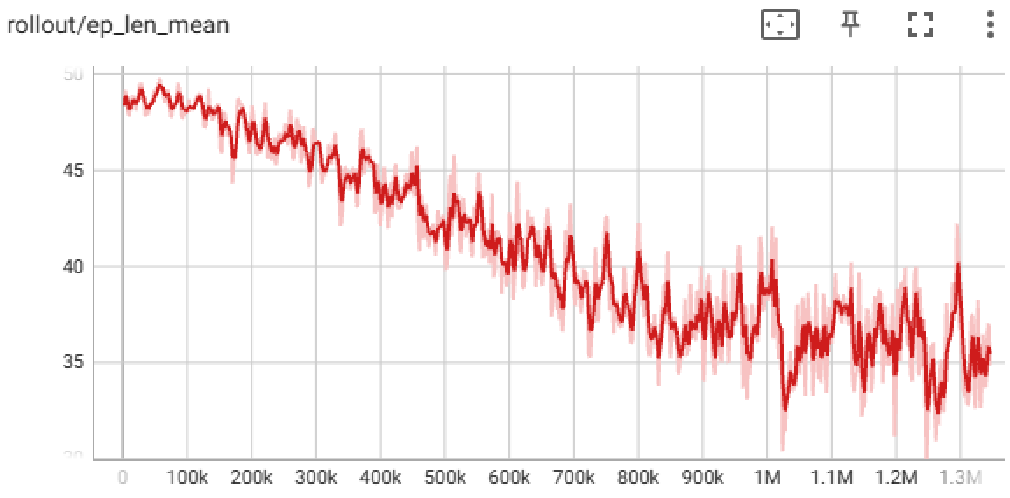


Figure (27): PPO 2-boxes episode reward mean

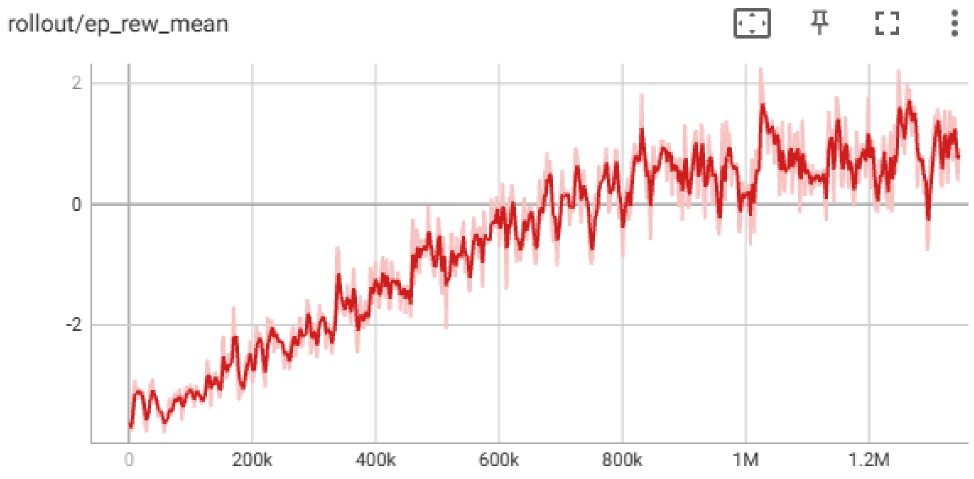
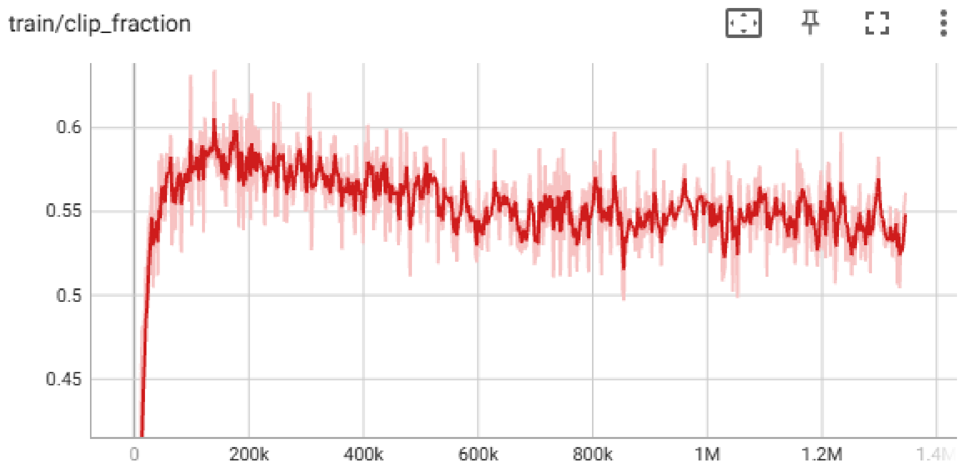


Figure (28): PPO 2-boxes clip fraction



Chapter 7
Results and Discussion

Chapter 7

Results and Discussion

This chapter presents the main results obtained from the experiments and discusses the performance of each reinforcement learning algorithm tested in the project.
It also analyzes how various parameters, reward structures, and hardware limitations affected the learning outcomes.

7.1 PPO Results

The PPO algorithm achieved stable and consistent learning only in the simplified 1-box environment [3].
After applying reward shaping and wrapper penalties, the model reached a success rate of about 90 % in most evaluation episodes.
The reward curves in TensorBoard showed clear improvement compared to earlier flat runs.

However, PPO struggled to generalize to 2-box (8×8) and 3-box (10×10) setups.
Training stopped improving after a certain reward threshold, and the model required long runtimes to recover.
Still, PPO demonstrated that reinforcement learning can solve Sokoban when the state space is controlled and reward signals are well-shaped.

7.2 DQN Results

The DQN algorithm produced encouraging results in both 1-box and 2-box environments [2].
During training, the reward trend showed continuous improvement, and the agent started solving nearly all 1-box training episodes efficiently.
For 2-box levels, DQN showed clear signs of learning (better than PPO) but evaluation could not be completed due to time constraints.

Overall, DQN training appeared more stable and less sensitive to hyper-parameter tuning.
Its performance suggested that with extended training and stronger hardware, DQN could outperform PPO for smaller Sokoban tasks.

7.3 Comparative Analysis

Table (7): Comparative Analysis

Feature	PPO	DQN
Learning Stability	High	Medium to High
Convergence Speed	Slow	Moderate
Resource Usage	Medium	High (but efficient)
1-Box Performance	Excellent	Excellent
2-Box Performance	Partial (evaluation)	Promising (training)
3-Box Performance	Unsuccessful	Unsuccessful
Sensitivity to Parameters	Very High	Moderate to High
Evaluation Completed	Yes	Partially

7.4 Observations

- Reward shaping and wrappers were the key factors behind the improvement in PPO.
- Hardware limitations (4 GB VRAM) significantly reduced training capacity and prevented large-scale experiments
- Internet and electricity instability caused interrupted training sessions, delaying results.
- PPO learned faster initially but plateaued quickly, while DQN improved steadily over time.
- Curriculum learning and HRL required far more time than available for this project.

Chapter 8
Conclusion and Future Works

Chapter 8
Conclusion and future works

8.1 Conclusion

This project explored the use of reinforcement learning (RL) techniques to train an intelligent agent to solve the Sokoban puzzle — a highly challenging environment that requires long-term planning and precise decision-making.

Over an eight-week development period, the team implemented, tested, and analyzed several RL algorithms, including PPO, HRL, and DQN.

By continuous testing, reward shaping, and environment adjustments, the agent successfully learned to solve 1-box environments with a high success rate and showed promising improvement for 2-boxes.

While PPO achieved strong results in simple configurations, DQN displayed more consistent improvement for more simple puzzles over time and appeared to be more stable under limited resources.

The experiments demonstrated that RL can be successfully applied to Sokoban puzzles, even when using modest hardware and within a restricted timeline.

The key outcomes of the project can be summarized as follows:

1. PPO model effectively learned with reward shaping and wrappers.
2. DQN showed stable learning progress and stability in training.
3. HRL and RPPO were limited by hardware and training time.
4. Reward shaping was essential to overcome sparse-reward challenges.
5. The final models provided a solid base for future enhancement.

8.2 Future Work

The current project achieved important progress, but several improvements can be made in future work to extend its capabilities and efficiency.

1. **Increase training time:** Allow models to run for longer timesteps to reach better convergence and higher success rates.
2. **Use stronger hardware:** Training on GPUs with larger VRAM would enable higher resolution environments and longer sequences.
3. **Enhance reward design:** Introduce adaptive reward shaping based on performance to guide learning more effectively.
4. **Implement curriculum learning:** Gradually increase difficulty from 1-box to multi-box levels automatically.
5. **Explore new algorithms:** Test advanced methods like SAC (Soft Actor-Critic) or DreamerV3 for better long-term planning.
6. **Add evaluation metrics:** Include quantitative comparisons such as episode length reduction, success ratio, and learning efficiency.
7. **Develop visualization tools:** Create dashboards that display agent behavior, reward evolution, and path efficiency.

References

IEEE ready preset from bibliography

[1]	S. Pastukhov ،""Solving Sokoban using Hierarchical Reinforcement Learning with Landmarks"،" arXiv ،] .2025Online .[Available: https://arxiv.org/abs/2504.04366.
[2]	M. Schrader ،"Gym-Sokoban Environment،" GitHub] .2024 ،Online .[Available: https://github.com/mpSchrader/gym-sokoban.
[3]	A. G. B. R. S. Sutton ،Reinforcement Learning: An Introduction ،Cambridge, MA: MIT Press .2018 ،
[4]	K. K. D. S. A. A. R. J. V. M. G. B. A. G. M. R. A. K. F. G. O. S. P. C. B. A. S. I. A. H. K. D. K. D. W. S. L. D. H. V. Mnih ،""Human-level control through deep reinforcement learning "،"*Nature* ،pp. 529-533 .2015 ،
[5]	F. W. P. D. A. R. O. K. J. Schulman ،""Proximal Policy Optimization Algorithms"،" arXiv] .2017 ،Online .[Available: https://arxiv.org/abs/1707.06347.
[6]	"Gymnasium Documentation،" Farama Foundation] .2024 ،Online .[Available: https://gymnasium.farama.org./
[7]	A. Raffin ،"Stable-Baselines3 Documentation،" GitHub] .2024 ،Online .[Available: https://stable-baselines3.readthedocs.io./
[8]	A. Raffin ،""Stable-Baselines3: Reliable Reinforcement Learning Implementations "،"*J. Mach. Learn. Res* ،. pp. 1-8 .2021 ،
[9]	"TensorBoard Guide،" TensorFlow] .2024 ،Online .[Available: https://www.tensorflow.org/tensorboard.

Manually written references

[1] A. G. Barto and R. S. Sutton, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[2] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, 2017.

[4] S. Pastukhov, "Solving Sokoban using Hierarchical Reinforcement Learning with Landmarks," *arXiv:2504.04366*, 2025.

[5] M. Schrader, "Gym-Sokoban Environment," GitHub, 2024. [Online]. Available: https://github.com/mpSchrader/gym-sokoban

[6] "Gymnasium Documentation," Farama Foundation, 2024. [Online]. Available: https://gymnasium.farama.org/

[7] A. Raffin, "Stable-Baselines3 Documentation," GitHub, 2024. [Online]. Available: https://stable-baselines3.readthedocs.io/

[8] A. Raffin et al., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021.

[9] "TensorBoard Guide," TensorFlow, 2024. [Online]. Available: https://www.tensorflow.org/tensorboard