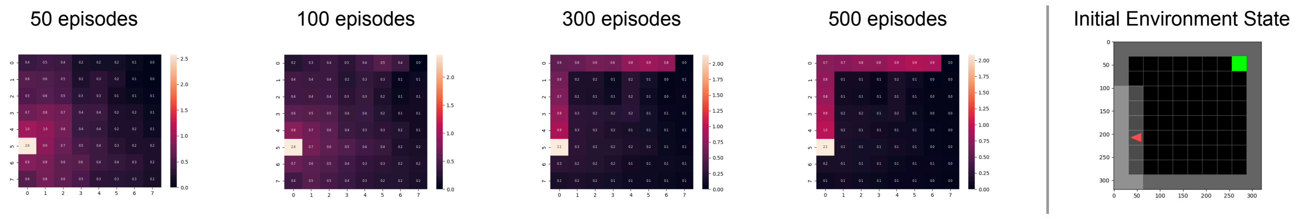
Reinforcement Learning - Final Project

This report examines solving the Minigrid game in three parts: Part A applies Q-Learning to the Empty and Key environments; Part B explores DQN variants, Policy Gradient, and Actor-Critic methods in the Empty environment; Part C addresses the Key environment using DQN. Each algorithm application includes a related Colab notebook, with five notebooks total.

General Approach

The biggest challenge was identifying the right hyperparameters. To address this, given the environment's variability, we fixed the environment during the hyperparameter search to prevent changes in artifacts location. This approach enabled us to analyze consistent, noise-free convergence graphs and observe the solution path becoming clearer with training progress by tracking the agent's visited locations. After identifying the optimal hyperparameters, we validated and, if necessary, adjusted them to the "stochastic" environment.

Heatmaps of Average Agent Visits across Episodes during Training



The heat maps show the agent's progress in the constant environment during training, with more frequently visited states highlighted in lighter shades.

Part A: Q-Learning

State Representation

we represent the game's fully observable, discrete environment using tuples based on the situation:
* **EMPTY Environment:** The state is depicted by a 3-entry tuples capturing the agent's position (column and row, ranging from 1 to 8) and direction (ranging from 0 to 3). This results in 8*8*4 = 256 distinct states. In the stochastic variation, where the goal's location changes, an additional dimension for the 3 possible goal's corner locations is included, expanding the total possibilities to 256*3.
* **KEY Environment:** This scenario extends the representation with two binary indicators to denote whether the key has been picked up and whether the door is opened, resulting in a 5-dimensional tuple (note that in this environment the goal location remains constant). For the stochastic variation, the tuple further incorporates the key's possible locations (8 rows across the 2 possible leftmost columns) and the door's location (8 possible rows, as the door remains in a consistent column), resulting in an 8-entry tuple. This comprehensive representation accounts for 8*8*4*2*2*8*2*8 = 131,072 possible states.

Action Representation and the Q-table

In the game's discrete action space, actions in the EMPTY environment range from 0 to 2 (3 total actions), while in the KEY environment, they range from 0 to 7 (8 actions, with two unused). The Q-table's dimensions are determined by the state space's dimensions and the number of actions. For instance, in the EMPTY environment with 8*8*4 possible states and 3 actions, the Q-table will have 8*8*4*3 entries, mapping all possible state-action combinations.

Experiments Design

In our experiments, we set fixed default values for hyperparameters known to lead to the agent's success and varied each one individually within a specific range to assess its impact.
We limited training to 100 steps per episode but adjusted the total number of episodes to ensure a comprehensive exploration of state-action combinations (Note: we could have terminated training earlier, but our goal was to determine how quickly the training process converges *under different hyperparameter values*, thus we didn’t opt for the lowest possible episode count).
We measured both scenarios where the artifacts in the environment are kept fixed in place across training episodes, which we refer to as a *"constant environment"*, and scenarios where the locations varied between episodes, which we refer to as *"stochastic environment"*.
All the parameters and their values are summarized in the following table:

	Empty Constant Environment	*Empty Stochastic Environment*	*Key Constant Environment*	*Key Stochastic Environment*
alpha	0.9	0.999	0.9	0.999
gamma	1	1	1	1
epsilon	1	1	1	1
epsilon decay	0.99	0.9995	0.99	0.9995
episode count	500	1000	500	36000

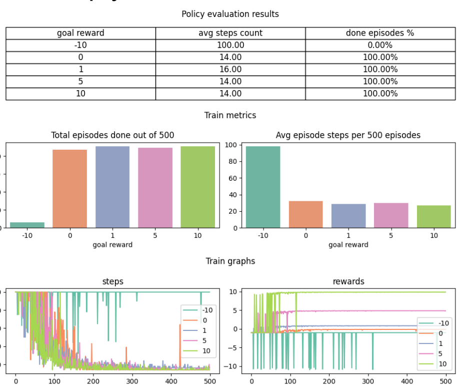
In the KEY environment, we assigned a reward of 1 for both picking up the key and opening the door for the first time. This hyperparameter was consistent across all experiments and was not further evaluated.
For each experiment, to assess the policy's **test-time** effectiveness, we tested the learned policy across 100 additional episodes. A summary table displays average steps per episode (max 100 due to step limit) and completion success rate. The training progress during **train-time** is monitored with a bar plot of solved episodes and their average steps, and a line plot tracking steps (up to 100), rewards per episode, and for Deep algorithms in the next chapter, the loss along the episodes.

Step and Goal reward

A positive or zero step reward encourages aimless exploration, while a negative goal reward discourages goal pursuit - both obstructing policy convergence. The results show that solution convergence is assured with any positive or zero goal reward, or a negative step reward, regardless of their exact value. These outcomes are expected and served primarily to validate that our implementation works as intended.

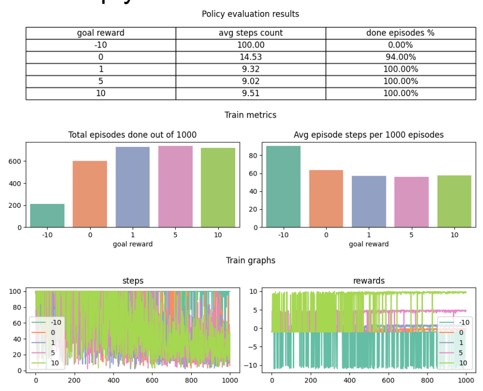
Comparison of train and test results across goal-reward values

Empty Constant Environment


Policy evaluation results

goal reward	avg steps count	done episodes %
-10	100.00	0.00%
0	14.00	100.00%
1	16.00	100.00%
5	14.00	100.00%
10	14.00	100.00%

Empty Stochastic Environment


Policy evaluation results

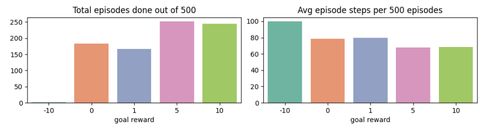
goal reward	avg steps count	done episodes %
-10	100.00	0.00%
0	14.53	94.00%
1	9.32	100.00%
5	9.02	100.00%
10	9.51	100.00%

Key Constant Environment

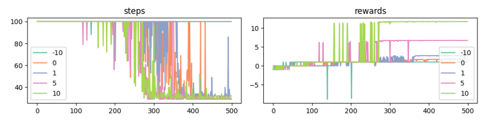
Policy evaluation results

goal reward	avg steps count	done episodes %
-10	100.00	0.00%
0	30.00	100.00%
1	32.00	100.00%
5	29.00	100.00%
10	30.00	100.00%

Train metrics



Train graphs

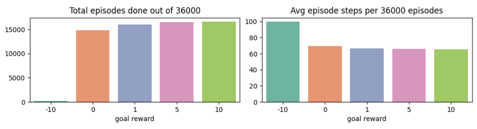


Key Stochastic Environment

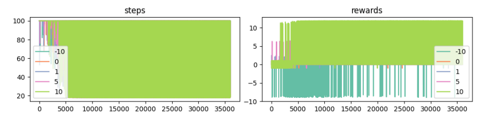
Policy evaluation results

goal reward	avg steps count	done episodes %
-10	100.00	0.00%
0	34.37	83.00%
1	41.29	75.00%
5	37.08	81.00%
10	43.21	73.00%

Train metrics



Train graphs



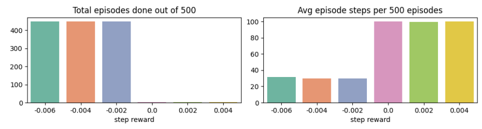
Comparison of train and test results across step-reward values

Empty Constant Environment

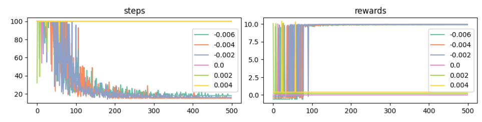
Policy evaluation results

step reward	avg steps count	done episodes %
-0.006	18.00	100.00%
-0.004	15.00	100.00%
-0.002	16.00	100.00%
0.0	100.00	0.00%
0.002	100.00	0.00%
0.004	100.00	0.00%

Train metrics



Train graphs

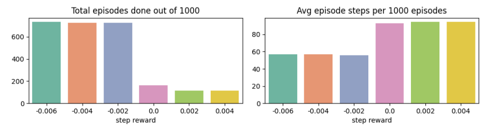


Empty Stochastic Environment

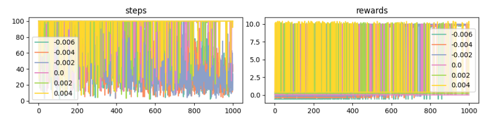
Policy evaluation results

step reward	avg steps count	done episodes %
-0.006	9.09	100.00%
-0.004	9.36	100.00%
-0.002	9.30	100.00%
0.0	100.00	0.00%
0.002	100.00	0.00%
0.004	100.00	0.00%

Train metrics



Train graphs

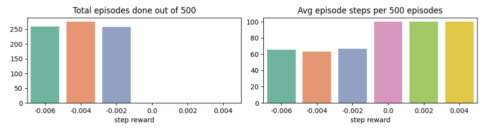


Key Constant Environment

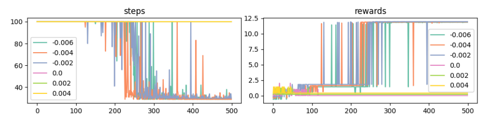
Policy evaluation results

step reward	avg steps count	done episodes %
-0.006	29.00	100.00%
-0.004	29.00	100.00%
-0.002	30.00	100.00%
0.0	100.00	0.00%
0.002	100.00	0.00%
0.004	100.00	0.00%

Train metrics



Train graphs

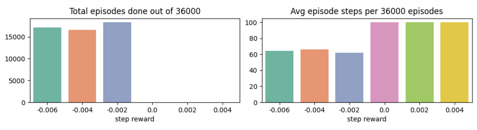


Key Stochastic Environment

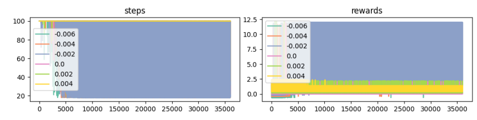
Policy evaluation results

step reward	avg steps count	done episodes %
-0.006	37.29	80.00%
-0.004	35.52	83.00%
-0.002	34.36	84.00%
0.0	100.00	0.00%
0.002	100.00	0.00%
0.004	100.00	0.00%

Train metrics



Train graphs



Alpha

The alpha hyperparameter, varying between 0 and 1, controls the learning rate of the agent—lower values lead to slower learning, while higher values accelerate it. Our observations indicate that changes in alpha did not substantially affect the algorithm's convergence in either setting, though the optimal scale varies: The constant environment allows for an alpha as low as 0.8, whereas the stochastic environment prefers higher alphas ranging from 0.9 to 1.

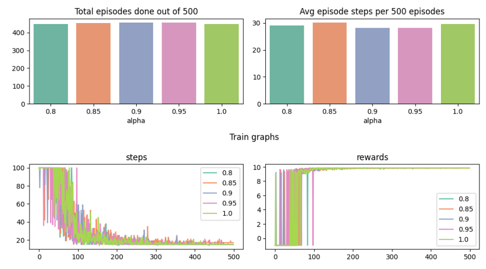
Comparison of train and test results across alpha values

Empty Constant Environment

Policy evaluation results

alpha	avg steps count	done episodes %
0.8	15.00	100.00%
0.85	17.00	100.00%
0.9	15.00	100.00%
0.95	15.00	100.00%
1.0	15.00	100.00%

Train metrics

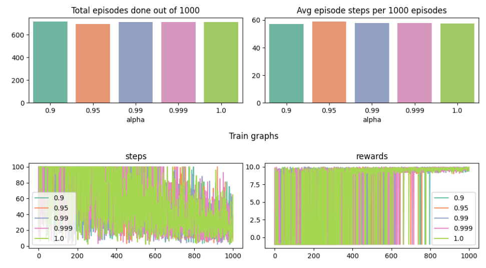


Empty Stochastic Environment

Policy evaluation results

alpha	avg steps count	done episodes %
0.9	9.11	100.00%
0.95	9.18	100.00%
0.99	9.35	100.00%
0.999	9.24	100.00%
1	8.83	100.00%

Train metrics

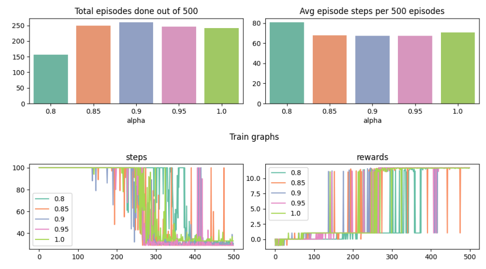


Key Constant Environment

Policy evaluation results

alpha	avg steps count	done episodes %
0.8	31.00	100.00%
0.85	29.00	100.00%
0.9	30.00	100.00%
0.95	29.00	100.00%
1.0	33.00	100.00%

Train metrics

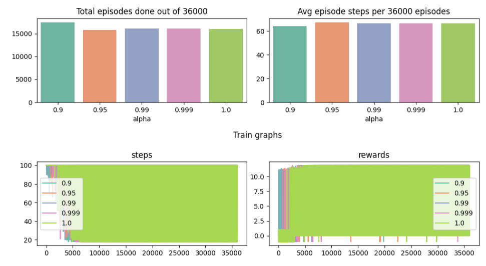


Key Stochastic Environment

Policy evaluation results

alpha	avg steps count	done episodes %
0.9	34.01	85.00%
0.95	35.13	83.00%
0.99	43.07	73.00%
0.999	35.90	83.00%
1	46.54	68.00%

Train metrics



Gamma

Gamma represents the weight given to future rewards compared to immediate rewards: A lower gamma value implies a greater emphasis on immediate rewards, while a higher gamma value suggests a stronger consideration of future rewards. The results show that in the EMPTY environment, the value of gamma is not critical, but in the KEY environment, gamma's value significantly affects outcomes. Specifically, higher Gamma values led to better results in both constant and stochastic settings. This could be attributed to the more complex nature of the rewards in the KEY environment compared to the EMPTY environment: In the KEY environment, the agent also receives rewards for picking up the key and for opening the door.

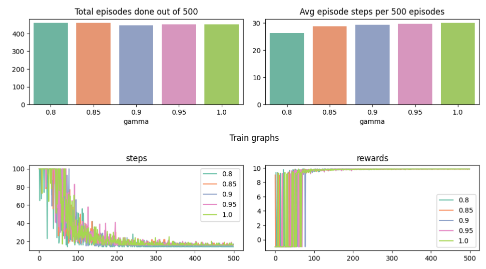
Comparison of train and test results across Gamma values

Empty Constant Environment

Policy evaluation results

gamma	avg steps count	done episodes %
0.8	14.00	100.00%
0.85	16.00	100.00%
0.9	15.00	100.00%
0.95	16.00	100.00%
1.0	16.00	100.00%

Train metrics

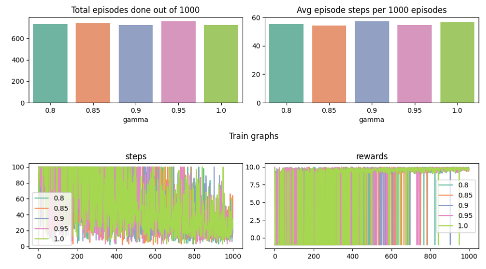


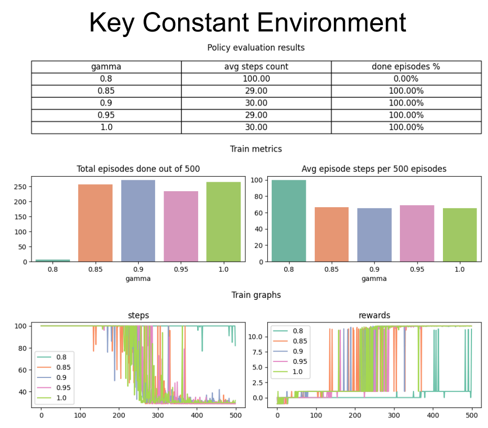
Empty Stochastic Environment

Policy evaluation results

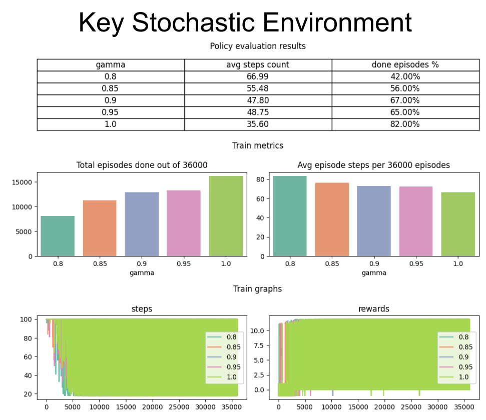
gamma	avg steps count	done episodes %
0.8	8.91	100.00%
0.85	9.27	100.00%
0.9	8.87	100.00%
0.95	9.36	100.00%
1.0	9.00	100.00%

Train metrics





gamma	avg steps count	done episodes %
0.8	100.00	0.00%
0.85	29.00	100.00%
0.9	30.00	100.00%
0.95	29.00	100.00%
1.0	30.00	100.00%

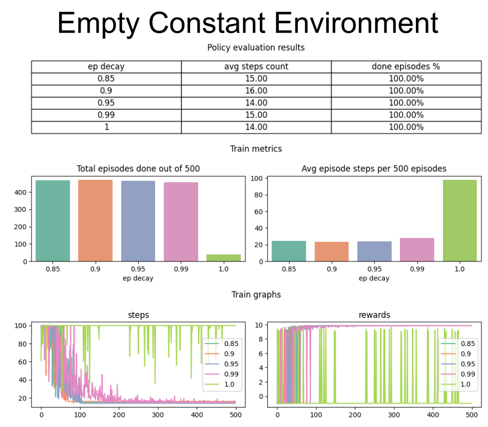


gamma	avg steps count	done episodes %
0.8	66.99	42.00%
0.85	55.48	56.00%
0.9	47.80	67.00%
0.95	48.75	65.00%
1.0	35.60	82.00%

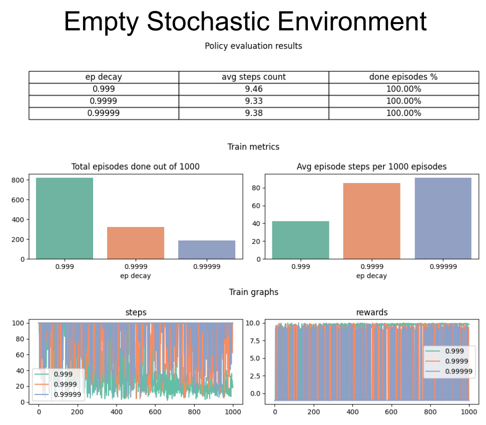
Epsilon & Epsilon Decay

Higher epsilon values promote exploration, making the agent more likely to choose random actions, whereas lower values encourage exploitation, with the agent preferring the best-known action.
Since the epsilon and epsilon decay hyperparameters are closely linked, we decided to set the initial epsilon value always to 1 and experimented with various *epsilon decay* rates. This approach means that higher epsilon decay rates encourage exploration, whereas lower rates speed up the move towards exploiting the established policy.
Among all hyperparameters, epsilon decay stands out as the most critical, necessitating tailored ranges for different environments (EMPTY and KEY). Generally, a preference for a high (but not too close to 1) epsilon decay rate is evident, ensuring adaptability without completely preventing change. Specifically, for the constant setting, an epsilon decay of 0.99 is preferred, while in the stochastic setting, 0.999 suits the EMPTY environment best and 0.9999 is optimal for the KEY environment.

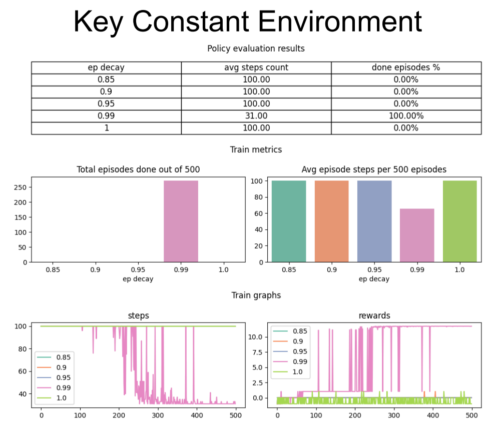
Comparison of train and test results across Epsilon-Dacy values



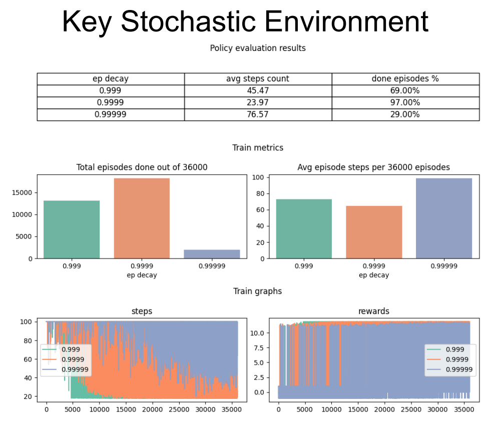
ep decay	avg steps count	done episodes %
0.85	15.00	100.00%
0.9	16.00	100.00%
0.95	14.00	100.00%
0.99	15.00	100.00%
1	14.00	100.00%



ep decay	avg steps count	done episodes %
0.999	9.46	100.00%
0.9999	9.33	100.00%
0.99999	9.38	100.00%



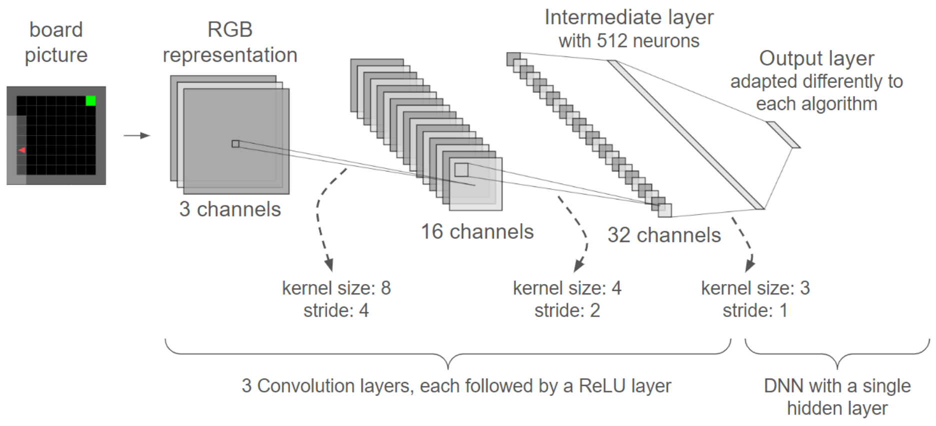
ep decay	avg steps count	done episodes %
0.85	100.00	0.00%
0.9	100.00	0.00%
0.95	100.00	0.00%
0.99	31.00	100.00%
1	100.00	0.00%



ep decay	avg steps count	done episodes %
0.999	45.47	69.00%
0.9999	23.97	97.00%
0.99999	76.57	29.00%

Part B: Deep Reinforcement Learning Solution for EMPTY Environment

This section focuses on solving the EMPTY environment using deep learning methods and comparing between them. For value-based algorithms, we examined Deep Q-Network (DQN) and its variations (Dueling DQN and Double DQN). For Policy Gradient algorithms, we examined the Proximal Policy Optimization (PPO) algorithm, and for the Actor-Critic type, we examined the Advantage Actor-Critic (A2C) algorithm.
In all algorithms, we used the same network structure: the image representing the state was processed through three consecutive convolutional layers, followed by a single dense layer with 512 nodes, which led to a final layer adapted to the different algorithms: In DQN and Double DQN, the last layer returns the Q-values for each possible action; In Dueling DQN, one head estimated the value of the current state, and the other head estimated the advantage for each action, then, the output combined both; In PPO and A2C, the state value alongside the probability for each action according to the learned policy in this state were returned.


board picture
RGB representation
3 channels
16 channels
32 channels
Intermediate layer with 512 neurons
Output layer adapted differently to each algorithm
kernel size: 8 stride: 4
kernel size: 4 stride: 2
kernel size: 3 stride: 1
3 Convolution layers, each followed by a ReLU layer
DNN with a single hidden layer

During the hyperparameter tuning process, we did not revisit environment-related hyperparameters (goal and step reward) as we did in the previous chapter, but focused solely on the hyperparameters relevant to the algorithms themselves. As before, to examine the hyperparameters' affect, we started with an initial configuration known to work and changed one parameter at a time. The following tables summarize the best configuration we found for each algorithm.

	DQN, DDQN & Dueling DQN	*PPO*	*A2C*
Learning rate	0.00005	0.00005	0.0001
Experience Replay Buffer Size	10,000	10,000	N/A
Batch Size (sampled from experience replay buffer)	32	32	N/A
Target Network Update Frequency (in episodes)	10	N/A	N/A
Policy Network Update Frequency (in steps)	1	1	200
Gamma	0.9	0.999	0.9
Epsilon	1	N/A	N/A
Epsilon Decay	0.999	N/A	N/A
Epsilon Clipping	N/A	0.2	N/A
Loss's Entropy Weight	N/A	N/A	0.1
Steps in episodes	100	100	100
Number of episodes - constant environment	500	500	1000
Number of episodes - Stochastic environment	1000	1000	1000

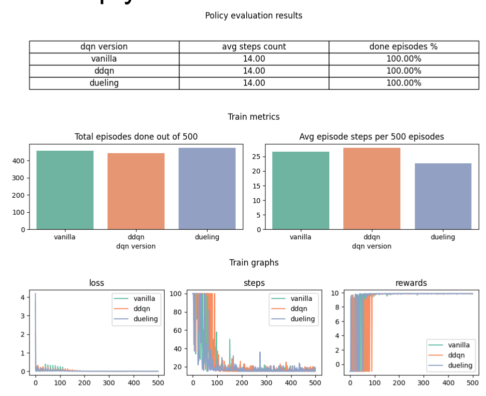
Like the previous chapter, we used more episodes than necessary to easily demonstrate on the graphs where training could have stopped and to mention what the sufficient number of episodes is.
As this section is lengthy, we'll only highlight the key findings from our experiments for each algorithm.
Additional graphs and examples, as well as further experiments and results, are available in the attached Colab notebook accompanying each algorithm's section.

DQN - Comparing Its Variations

In the constant environment, the Dueling DQN achieved the best results, as it led to a policy that solved the environment in the fewest steps most quickly during training. In the stochastic environment, DDQN was preferred for the same reasons. Additionally, it was the only one to succeed in 100% of the evaluations during test time. Overall, the difference between the three variations was minimal, and it appears that all achieved good results in solving the environment. Further details on hyperparameter tuning (gamma, learning rate and epsilon decay) are available in the Colab notebook.

Comparison of DQN algorithms - Empty Environment

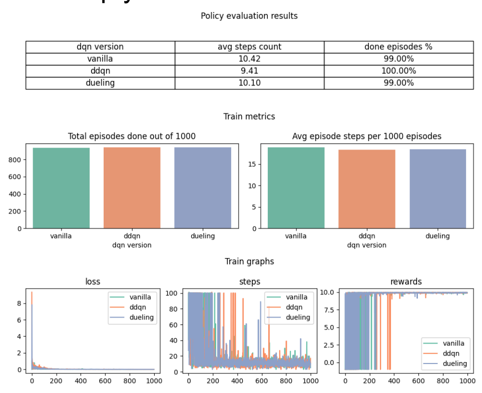
Empty Constant Environment



Policy evaluation results

dqn version	avg steps count	done episodes %
vanilla	14.00	100.00%
ddqn	14.00	100.00%
dueling	14.00	100.00%

Empty Stochastic Environment



Policy evaluation results

dqn version	avg steps count	done episodes %
vanilla	10.42	99.00%
ddqn	9.41	100.00%
dueling	10.10	99.00%

The PPO algorithm

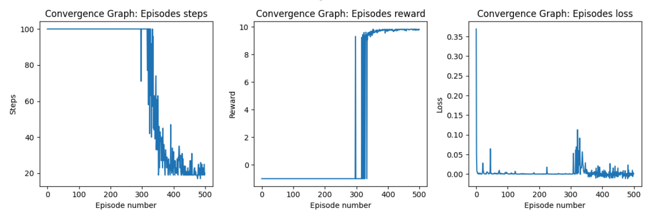
PPO was the slowest to converge, requiring more than 300 episodes in the constant environment to converge, and only beginning to do so near the 600th episode in the stochastic environment. It's important to note that PPO doesn't always complete games with the optimal step count like DQN because, unlike DQN's greedy approach, PPO keeps a stochastic element in its policy, choosing actions based on its learned distribution. Further details on hyperparameter tuning (gamma, learning rate and epsilon clipping) are available in the Colab notebook.

PPO algorithms - Empty Environment

Empty Constant Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 19.35
- Environment Solved in 100.00% of Trials

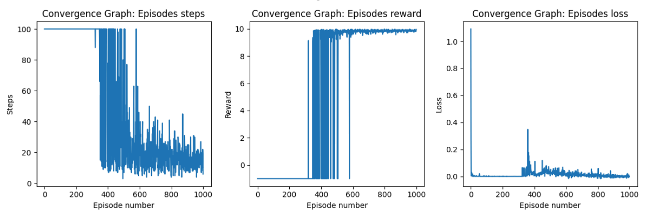
Train graphs



Empty Stochastic Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 12.02
- Environment Solved in 100.00% of Trials

Train graphs



The A2C algorithm [Colab Notebook Link]

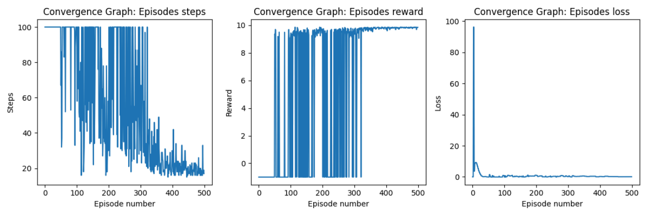
In the constant environment, the algorithm converged after roughly 300 episodes; in the stochastic environment, convergence required nearly all 1000 episodes, as clearly indicated by the reward graph. But, in terms of runtime, this algorithm was the shortest because it trains every 200 steps (equivalent to up to 2 consecutive episodes), unlike DQN and PPO, which train every step. This made it more convenient to tune, despite being less stable than the others.

A2C algorithms - Empty Environment

Empty Constant Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 17.64
- Environment Solved in 100.00% of Trials

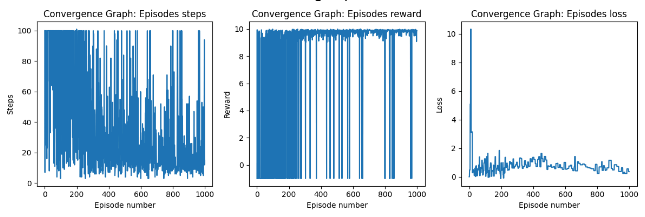
Train graphs



Empty Stochastic Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 26.68
- Environment Solved in 94.00% of Trials

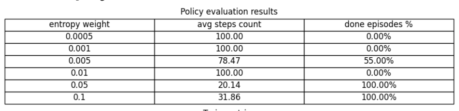
Train graphs



A critical hyperparameter identified in our experiments with A2C is the "Epsilon Weight", which balances exploration and exploitation by penalizing uniform action distributions. Comparing different epsilon weight values and analyzing agent visitation plots in the constant environment shows that higher values enhance exploration, enabling goal discovery and policy optimization; In contrast, low values cause quick convergence to a static policy, potentially trapping the agent in suboptimal states and hindering goal location, thereby increasing the likelihood of failure.

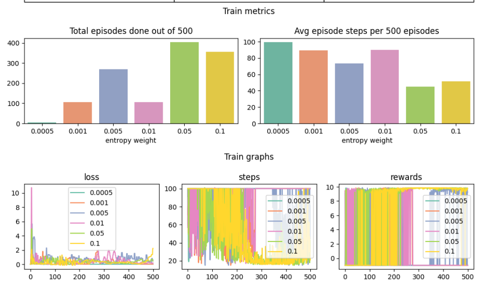
Comparison of Different Loss's Entropy Weight

Empty Constant Environment

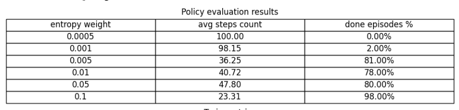


Policy evaluation results

entropy weight	avg steps count	done episodes %
0.0005	100.00	0.00%
0.001	100.00	0.00%
0.005	78.47	55.00%
0.01	100.00	0.00%
0.05	20.14	100.00%
0.1	31.86	100.00%

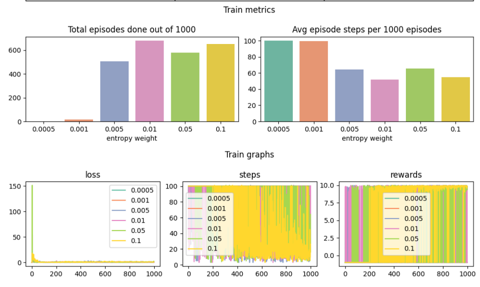


Empty Stochastic Environment

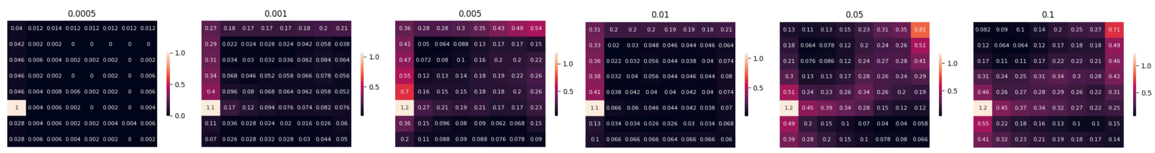


Policy evaluation results

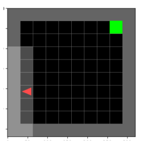
entropy weight	avg steps count	done episodes %
0.0005	100.00	0.00%
0.001	98.15	2.00%
0.005	36.25	81.00%
0.01	40.72	78.00%
0.05	47.80	80.00%
0.1	23.31	98.00%



The heat maps illustrate the agent's progress in finding a solution over the course of training with different entropy weights in the constant environment. States that are visited more frequently are highlighted in lighter shades. We can see that the lower the value of the entropy loss, the more likely it is that the agent remains at its place and explores less.



Initial Env State:



Further details on hyperparameter tuning (gamma and learning rate) are available in the Colab notebook.

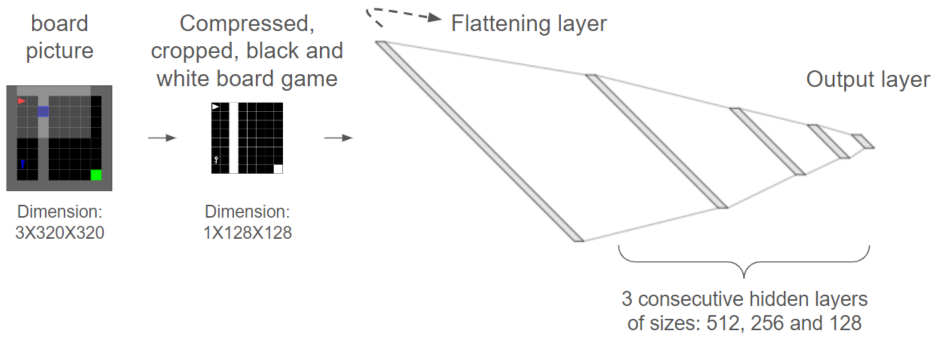
Summary: Tabular vs. Deep Learning in the EMPTY Environment

When comparing Q-learning to the deep learning algorithms (DQN, PPO, A2C), we find that the tabular method is more effective and converges quicker in the constant environment. However, in the stochastic environment, deep learning methods both outperform and converge faster because they can approximate the optimal actions through continuous function evaluation.This capability is essential when the agent comes across unseen states during testing, a challenge where tabular methods fail.

Part C: DQN Solution for KEY Environment

We chose the DQN algorithm to solve the KEY environment. Since using the same approach as in the previous part led to lengthy training times and failed to converge, we modified both the network structure and the reward reshaping.

In terms of **network structure**, instead of using convolutional networks, we preprocessed the state's image by cropping unnecessary borders, converting it to black and white, and compressing it to a 128x128 resolution. To prevent redundant processing, we implemented hashing to efficiently retrieve and reuse images of previously encountered states.


board picture
Dimension: 3X320X320
Compressed, cropped, black and white board game
Dimension: 1X128X128
Flattening layer
Output layer
3 consecutive hidden layers of sizes: 512, 256 and 128

For **reward reshaping**, we recalibrated the reward scale: the agent now earns five points for actions like picking up a key or opening a door for the first time, and ten points for reaching the goal. Failed toggle or pickup actions incur a five-point penalty, preventing such unnecessary actions. Moreover, the agent receives a negative reward proportional to the Manhattan distance to artifact locations (key/door/goal), normalized by dividing by 10, to encourage navigation toward them.

Like before, we tested our hyperparameters on a constant environment where the artifacts remain fixed, and after finding these, we tuned them for the stochastic environment where locations change upon reset. The optimal hyperparameter values are detailed in the following table.

	Key Constant Environment	*Key Stochastic Environment*
Learning rate	0.0005	0.0005
Experience Replay Buffer Size	10,000	10,000
Batch Size (sampled from experience replay buffer)	32	32
Target Network Update Frequency (in episodes)	10	10
Policy Network Update Frequency (in steps)	1	1
Gamma	0.9	0.9
Epsilon	1	1
Epsilon Decay	0.99995	0.99995
Steps in episodes	100	100
Number of episodes	500	2000

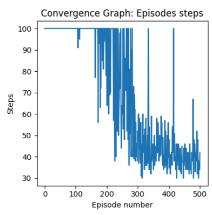
And the results are as follows:

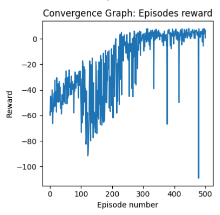
DQN algorithms - KEY Environment

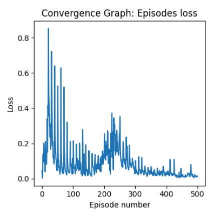
Empty Constant Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 29.00
- Environment Solved in 100.00% of Trials

Train graphs



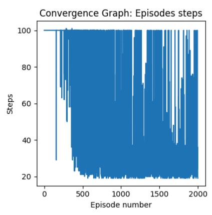


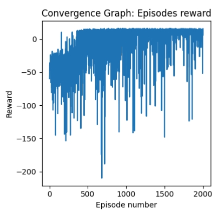


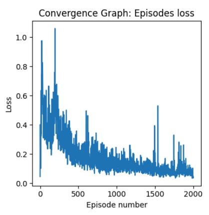
Empty Stochastic Environment

Policy Evaluation Summary (over 100 trials):
- Average Steps Taken: 34.14
- Environment Solved in 83.00% of Trials

Train graphs

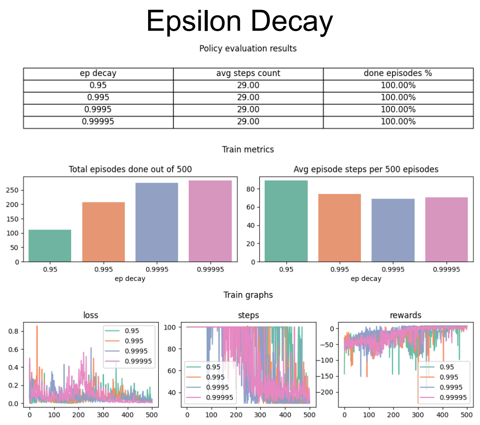






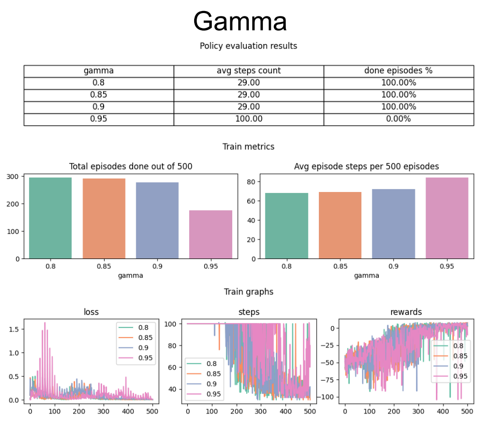
From comparing different hyperparameters values in the constant environment, it becomes apparent that certain gamma and epsilon decay values demonstrate distinct advantages: **A higher epsilon decay** is preferred, likely because it encourages exploration, which is crucial for complex environments and enables finding the key, the door, and the goal before the agent starts to strictly follow the learned policy due to the exploitation trade-off. This also aligns with Part A's results, where the Key environment was examined using the Q-Learning algorithm.
A **moderate gamma value** is favored as it allows the agent to value immediate rewards (such as those given when acquiring the key or opening the door) and not only take into account the reward for reaching the goal.

Comparison of Different Gamma and Epsilon Decay Values in the Constant Environment


Epsilon Decay

Policy evaluation results

ep decay	avg steps count	done episodes %
0.95	29.00	100.00%
0.995	29.00	100.00%
0.9995	29.00	100.00%
0.99995	29.00	100.00%


Gamma

Policy evaluation results

gamma	avg steps count	done episodes %
0.8	29.00	100.00%
0.85	29.00	100.00%
0.9	29.00	100.00%
0.95	100.00	0.00%

Further details on hyperparameter tuning (learning rate and the different DQN versions - DDQN, Dueling, and vanilla DQN) are available in the Colab notebook.

Conclusion

The project aimed to solve the MiniGrid game using both tabular and deep reinforcement learning methods, including both value-based and policy-based approaches. As mentioned at the beginning of the report, the greatest challenge was finding the optimal hyperparameters that would ensure convergence: While tabular methods had relatively short training runtimes, deep learning methods, which involve complex networks, required significantly longer runtimes, adding complexity to the hyperparameters tuning process. However, as noted before, deep learning methods have a significant advantage over tabular methods as they can generalize the agent's policy to states not encountered during training, a capability that tabular methods lack.