Universiteit
Leiden
The Netherlands

Data Science and Artificial Intelligence

The exploration-exploitation trade-off in Reinforcement Learning:
an EduGym implementation

Matteo Dröge

Supervisors:
Aske Plaat
Koen Ponse

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

20/07/2024

Abstract

This thesis assesses the exploration-exploitation trade-off in reinforcement learning and examines how this concept could most suitably be implemented for EduGym. The exploration-exploitation trade-off is a fundamental concept within reinforcement learning. It is of high importance that students new to this field understand such concepts sufficiently. The reality however, is that students often struggle with bringing such concepts from theory to practice. EduGym, an interactive suite for reinforcement learning teaching helps to overcome this problem. By providing a theoretical framework on the trade-off and constructing experiments that bring this trade-off to practice, this thesis proposes a constructive approach as to how this trade-off could be best showcased using EduGym. This results in an implementation consisting of a Q-Learning agent alongside a maze-like environment called the Mouse Maze. This environment features a mouse as agent, as well as cheeses and traps. Experiments are performed on variants of this environment, which grow in complexity. Ultimately, this proved to demonstrate how several reinforcement learning aspects, such as state dimensionality, environment complexity and reward feedback, affect the exploration-exploitation trade-off. The experiment results highlight those aspects with the use of graphs and heat-maps, making it better to understand for students how this trade-off unfolds in practice.

Contents

1 Introduction

Reinforcement learning is one of the most interesting fields within artificial intelligence. It is a type of machine learning based on trial-and-error learning, inspired by how humans and animals learn from feedback in the pursuit of achieving goals [KLM95]. The key concept of reinforcement learning is to model an implementation of an agent within an environment [SB18]. Here, the agent will interact with the environment by taking actions, based on which it will receive feedback in the form of rewards. This way, the agent learns by adjusting its strategy accordingly to learn the optimal behaviour through exploration and exploitation.

Over the years, the underlying techniques within reinforcement learning have improved a lot. These developments, however, mostly stem from research conducted by Sutton and Barto [SB99] into now widely known algorithms such as Q-Learning[Wat89]. One of the main challenges within reinforcement learning, and models like Q-Learning, remains the optimal balance between exploration and exploitation [Fru19]. The exploration-exploitation trade-off refers to the dilemma of choosing between exploring new options (exploration) and exploiting known options to maximise immediate rewards (exploitation). Balancing exploration and exploitation is therefore crucial for effective learning and decision-making.

Especially for people who are new to reinforcement learning, the exploration-exploitation trade-off is a fundamental concept that must be understood well. EduGym [MMBY+23a] is an interactive suite for reinforcement learning education, intended for an audience new to reinforcement learning. While still being in development, it aims to serve as an environment to educate this audience by showcasing key concepts within reinforcement learning by the hand of interactive notebooks.

In this thesis, an assessment on the exploration-exploitation trade-off will be made, alongside a thorough examination of how this could most suitably be implemented and demonstrated using EduGym.

1.1 Problem statement

As stated, the exploration-exploitation trade-off remains one of the main challenges within reinforcement learning. Therefore, it is especially crucial for people new to reinforcement learning to understand its importance sufficiently. This thesis aims to emphasise the importance of the exploration-exploitation trade-off both theoretically and in practice, finally providing a suitable implementation for EduGym to showcase its importance. The two main research questions that are the motivation behind this thesis are:

- What is the importance of the exploration-exploitation trade-off in reinforcement learning?
- What is the best way to implement and showcase the effects of this trade-off in EduGym?

In order to give a comprehensive answer to these two research questions, a theoretical framework on the exploration-exploitation trade-off will be provided, alongside multiple structured experiments that showcase this trade-off in practice. Here, an implementation of a Q-Learning agent will be utilised, which will be trained on multiple instances of an environment that will gradually increase

in complexity, in order to fully demonstrate how different situations are affected by the trade-off. The performed experiments will therefore guide as a constructive approach as to how an implementation showcasing the trade-off could be most suitably shaped. Based on the results of these experiments, a conclusion will be made that takes in regard all of the important insights that come to the surface. Finally, the insights gained will help in providing the end goal of this thesis: a working implementation of an environment for EduGym.

1.2 Thesis overview

To start off with in Section 2, a selection of related work will be examined. This includes an assessment on work related to the exploration-exploitation trade-off, as well as what a Markov Decision Process is, alongside the Gymnasium library and how this relates to EduGym. In Section 3, underlying principles behind reinforcement learning will be provided. This includes definitions of both the modelled Q-Learning algorithm (the agent) as well as the general elements that are related to the environments it will be utilised on. Following that in Section 4, the set-up of multiple experiments will be discussed. The results of these experiments will be reviewed in Section 5. Finally in Section 6, conclusions will be drawn based on these results, including a final assessment on how these results gave insights as to how the trade-off could be most suitably implemented for EduGym.

2 Related Work

Within reinforcement learning, the goal is to implement an agent that will learn intelligent or 'good' behaviour through experience collected by traversing a certain environment. The problem is often formalised as a process, where the agent will gain experience through rewards obtained from the environment, with the goal to maximise the cumulative reward over a long period of time.

2.1 Markov Decision Process

The environment of a reinforcement learning problem is traditionally formulated as a Markov Decision Process [GR13]. This type of environment enables the modelled agent to get a suitable sense of the state of the environment it is in, so that it will be able to take actions accordingly. Markov Decision Processes (MDPs) are intended to include all necessary aspects in order to the describe these decision-making situations. This is done by describing the agent's current state, its possible actions, the reward and the observed transition in changing states. Formally, an MDP thus consists of the following components:

- $s \in S$: representing all possible states in the environment
- $a \in A$: representing all possible actions that can be performed within the environment
- $P(s'|s, a)$: the transition probability from the current to the next state
- $R(s, a)$: the reward that will be obtained when reaching a state s via action a

When the agent decides to take a certain action given a certain state, it receives a reward and moves to the next state within the environment. This process is visualised in Figure 1 below. By definition, this process satisfies the Markov property, meaning that future events rely solely on the present state and chosen action, not relying on the history of events that happened before. This makes it possible for the model to learn based on the obtained rewards without keeping track of what it is actually doing, considerably simplifying the problem.

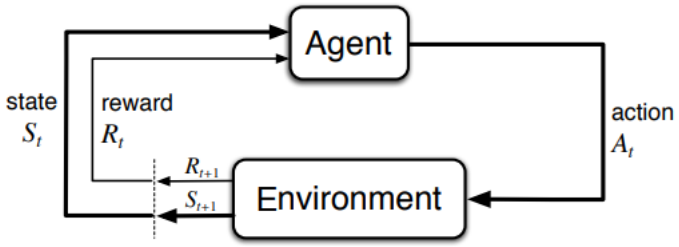


Figure 1: Graphical representation of a Markov Decision Process. Note that the present and future of states, actions and rewards are clarified using time-step indicators t and t_{+1} within the process [SB18].

2.2 Exploration vs. Exploitation

In order for the agent to properly learn from the environment it is situated in, the agent needs to update its behaviour in order to make better decisions. Because of the Markov property of the environment, future observations are impacted by any present decision the agent makes. Therefore, the decision-making process of the agent is crucial for its learning process and the resulting updates in its behaviour. The agent needs to learn from experience by traversing many of the possible states within the environment, while also taking into account to maximise its cumulative reward over a longer period of time. This is where the exploration-exploitation trade-off arises [Fru19], where two opposing strategies in action selection conflict with each other, namely:

Exploration: choosing new, unknown options within the environment to gain more experience, allowing for better future decisions.

Exploitation: choosing the best option based on the current knowledge, in order to maximise immediate rewards.

While both strategies have their importance, they also come with negative consequences when applied improperly. Exploring new options may come at a high cost, especially when done too frequent. This could have numerous negative outcomes, such as a low or minimal cumulative reward, as well as worsening the learning process of the agent. On the other hand, enabling the agent to exploit options on the current knowledge too frequent could potentially lead to sub-optimal behaviour, since the known knowledge does not necessarily include all needed knowledge in order to obtain the optimal result. These opposing strategies result in a dilemma, with the challenge being to find an optimal balance between both exploration and exploitation in order for the agent to thrive within a certain environment.

In order to solve this trade-off, a lot of research has been conducted. The first studies into this dilemma actually precede reinforcement learning [Tho33], when it was first studied in a simplified case of multi-armed bandits [VBW15]. Thompson sampling is an algorithm where actions are taken sequentially based on a probability distribution in order to balance both exploration and exploitation. In cases such as bandit problems, like Bernoulli bandits, this proves to be very effective.

Thompson sampling has gained significantly more interest following the developments in reinforcement learning. Multiple adaptations on Thompson sampling have now successfully been applied to a variety of domains [RRK+17], including reinforcement learning and MDPs [OGNJ17] in order to tackle the exploration-exploitation trade-off. Besides this, a lot of research has been conducted in order to optimise the policies of reinforcement learning algorithms [Fru19]. Proposals for new improved algorithms, such as for adaptive opportunistic bandits [WGL17] or the usage of confidence bounds [Aue02], are all aiming to deal with the exploration-exploitation trade-off in reinforcement learning problems.

2.3 Gymnasium and EduGym

Gymnasium [TTK+24] intends to represent general reinforcement learning problems via an interactive manner. Being a fork of OpenAI's Gym library [BCP+16], it provides a framework consisting of a large collection of environments with a common, simple interface. This allows for the user to make use of pre-existing environments and get used to certain reinforcement learning problems in a convenient and accessible manner. While textbook teaching material such as Sutton & Barto [SB18] are sufficient for the theoretical explanation behind reinforcement learning principles, students often struggle with translating this into actual code. Without practical examples, students may endure a longer learning cycle, making Gymnasium a potential solution to this problem.

Documented codebases such as Gymnasium are helpful, but also come with difficulties for students new to this field. Many codebases, which like Gymnasium are mostly based on the Gym paradigm, are research oriented. They often incorporate state-of-the-art algorithms which combine complicated principles, as well as being implemented in high-dimensional environments. Even though Gymnasium itself tries to overcome this problem by implementing simple, low-dimensional problems as well, it is still not ideal for education. Most provided environments combine several types of challenges, making it hard for students to distinguish them.

EduGym [MMBY+23a] intends to be an interactive companion to other reinforcement learning teaching material. While still being in development, it aims to teach students particular challenges within reinforcement learning based on the Gymnasium framework. It contains a range of environments designed that each highlight one of the numerous challenges within reinforcement learning. Alongside these environments, interactive notebooks are provided which illustrate the challenges [MMBY+23b]. Here, the student will be guided through the environment by explaining its challenge, covering possible approaches and illustrating the performances.

3 Methods

In this section, definitions of the Q-Learning agent as well as the general elements that exist within the environments will be explained.

3.1 Design of the Environments

To design both practical and useful environments to showcase the trade-off in, it must obey the principles of a Markov Decision Process. This means a stochastic environment, that can give the correct feedback to the agent through states and rewards, in which the agent can perform an action based on a determined set of actions. Besides these principles, it is also important to keep in mind that the goal is to present these environments in such a way that they are easily understandable when presented through EduGym. This calls for a simple approach, where a two-dimensional environment would be best suited. The general idea of the visualisation of the environment is a maze, where the agent is visualised as a mouse and traverses the environment towards a certain exit. On its way, it may encounter items represented as cheeses as well potential traps, both of which have their own unique rewards. An example of the Mouse Maze environment is given in Figure 2. Formally, all elements are listed on the following page.

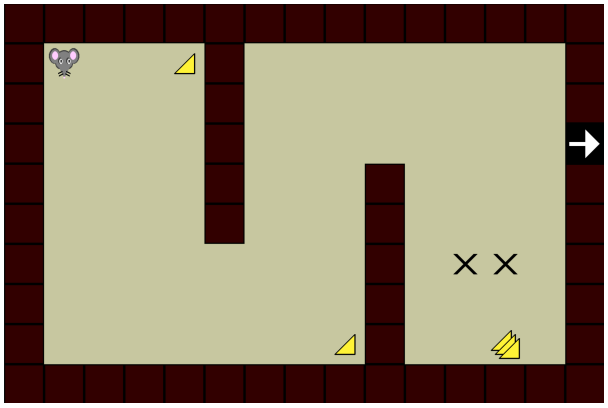


Figure 2: Example of a Mouse Maze environment featuring a mouse as agent, cheeses and traps.

state: a state s is the current situation of the environment.

state space: the state space of all possible situations of the environment. Here, each state $s \in S$ represents a single state within the state space. The state space is based on the size of the environment, as well as the number of items that are present within it.

action: an action a is a move that can be performed by the agent within the environment.

action space: the action space consists of all possible actions $a \in A$, being the set of $A =$[up, down, left, right].

agent: the agent, represented as a mouse, traverses through the environment's states by performing actions.

step: a step is the execution of a single action by the agent, based on which it will move itself within the environment. The current state s will be updated to the next state s'. Based on the properties of s', the agent will receive a reward.

reward: a reward R is a form of feedback obtained by the agent when a step has been performed. Initially, each step will result in a reward of -1.

wall: a wall is an object in the environment that is impassable. When the agent tries to move to a state containing a wall, the agent will still perform the chosen action and receive a reward. However, the agent will remain in the exact same state, in this case meaning that $s' = s$.

cheese item: an item c that is able to be collected. This generally results in an additional reward of $+15$.

cheese pile item: a more valuable cheese item. When picked up, this generally results in an additional reward of $+50$.

trap: when the agent encounters a trap state, it will receive an additional reward of -100. The agent will also be reset to its starting point in the environment.

time-steps: a value that determines how many steps will be performed by the algorithm within the environment. More time-steps means a longer period for the algorithm to learn. A time-step t denotes the current step in this process.

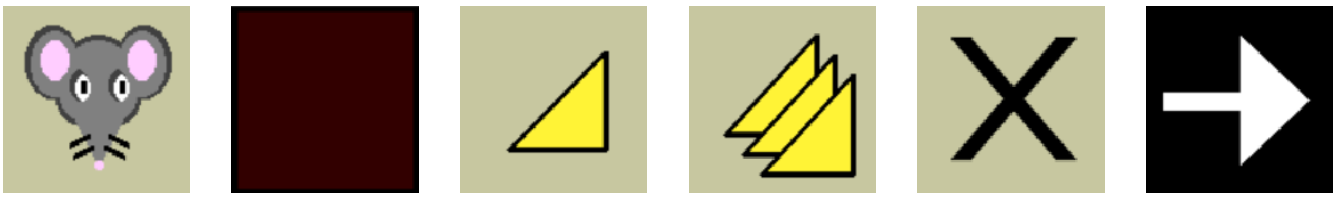
episode: a formal time frame from the initial starting point of the environment until the agent reaches an end state. Multiple episodes can occur within the set amount of time-steps.

exit: a final state, ending an episode. The obtained reward when moving into the exit will generally be 0. A new episode will then begin, resetting all elements within the environment to its starting position.

3.2 Visualisation

Since the environment has to be implemented in a way that it is both interactive and clear to understand, it is important that the visual aspects have a clear correspondence to the elements they represent. In Figure 3 below, the elements that are visible within the Mouse Maze environment and what they represent are shown. All elements are visualised using Pygame, a Python library utilised by EduGym to implement their interactive environments. In the case for this implementation a simplistic approach was used, aiming to be as straightforward as possible for the user, while maintaining all characteristics that are bound to the elements they represent. By doing so, it will allow the user to understand the environment promptly and shift their focus on understanding the underlying principles that this environment show-cases.

The agent has been drawn in more detail, since this is the most important element within the interactive environment. The user is able to play and navigate itself through the environment by moving the agent. The wall object is drawn in a simple manner, indicating a space that is impassable. The cheese and cheese pile items are drawn as yellow triangles, with the cheese pile object consisting of three cheeses indicating a higher reward. An 'X' is drawn for the trap, a sign generally interpreted having a negative semantic. Finally, the exit is drawn as a clear arrow, indicating a way out.



(a) Agent (b) Wall (c) Cheese (d) Cheese pile (e) Trap (f) Exit

Figure 3: Visual representation of the elements within the environments.

3.3 Q-Learning agent

In principle, Q-Learning is a simple approach for agents to learn how to act optimally in Markovian domains. Originally presented by Watkins in 1989 [Wat89], it is an incremental algorithm for estimating an optimal decision strategy, which is now widely used in reinforcement learning [CL20].

3.3.1 Formalism

As stated, the algorithm learns through experience. One of the most important aspects of this algorithm is keeping track of the action values in a Q-table, formally written as $Q(s, a)$. Here, an estimated value for action a in state s will be stored. The Q-table can be initialised arbitrarily and will be updated after each step. This is essentially how the algorithm stores its experience, which will be utilised when deciding what action to select. To do so, it follows a certain policy π, which is where the exploration and exploitation trade-off mostly revolves around in this case.

The ϵ-greedy action selection method intends to realise this in a simple manner. By determining a certain ϵ value beforehand, the algorithm will be instructed to either explore or exploit based on this value. The ϵ value lies between 0 and 1 and will be compared to a randomly chosen value within those boundaries in each step of the process. When the random chosen value is higher than the ϵ value, the algorithm will exploit in this state. In this case, this would be choosing the best action given the highest action value found in the Q-table for the current state. Otherwise, the algorithm will be instructed to explore, meaning that a random action will be chosen. In other words, the value of the ϵ parameter can be seen as the probability that the algorithm will explore for each step in the process. Formally, this selection method is defined as follows:

$$\pi_{\epsilon-greedy}(a) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_{b \in A} Q(b) \\ \frac{\epsilon}{(|A|-1)}, & \text{otherwise} \end{cases}$$

When an action is chosen, it will be performed and the algorithm will receive feedback in the form of a reward based on the next state it will move to. Following this reward it will update the Q-table based on an update rule, defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(s_t, a_t)]$$

Here, the state-action values are updated based on multiple components. This includes the current value for the state and performed action in $Q(s_t, a_t)$, the obtained reward R_{t+1} from the next state and the highest action value from the next state $\max_a Q(S_{t+1}, a)$. Alongside this, two hyper-parameters are included in this formula. The α parameter corresponds to the learning rate, effectively determining the rate at which the Q-table will be updated based on the obtained reward and upcoming state-action values. A higher learning rate will correspond to quicker adaptation. The discount factor γ determines the influence of the next states highest action value. Both parameters are initialised between 0 and 1, like the ϵ parameter.

3.3.2 Implementation

The Q-Learning agent as well as the environment are implemented using Python. The pseudo-code which describes how the algorithm works in principle is visualised below in Algorithm 1. To begin with, all values in the Q-table will be initialised arbitrarily. In the used implementation, these values will all be set to 0. Then, the algorithm will start and begin a loop. For each new episode, the current state will be initialised as a starting point. Following that for each step, an action will be chosen based on the policy, being ϵ-greedy in this case. The chosen action will be executed and the reward and next state will be observed. The update rule will then update the Q-table based on these observations. Finally, the current state will be set to the next state. This all continues in a loop until a terminal state is reached, resetting the environment and commencing a new episode.

In practice, the algorithm is not looping over episodes itself, but rather runs continuously, restarting the environment when a terminal state is reached and runs until the maximum number of allowed time-steps is reached. In this case, the number of total episodes is undefined and is related to the performance of the algorithm, as well as the total amount of time-steps the algorithm is allowed to run. The number of completed episodes can be used to measure the algorithm's performance, but given the main focus of this thesis, this will not be done.

Algorithm 1 Q-Learning

Initialise $Q(s, a)$ for all $s \in S$ and all $a \in A$ arbitrarily
repeat for each episode:
    Initialise s
    loop for each step of episode:
        Choose a from s using policy derived from Q (ϵ-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
        $s \leftarrow s'$
    until s is terminal
until end

Algorithm 1: Pseudo-code of the Q-Learning algorithm as implemented using Python. Note that apostrophes are used to denote current and next states (s'), instead of time-steps.

4 Experiments

The following experiments are meant to highlight certain aspects that are of influence in the exploration-exploitation trade-off. The first four experiments are based on an environment that grows in complexity. Starting off from a small, empty environment, experiments will be conducted where in each experiment a new feature will be added to the environment, such as an increase in state space, as well as the addition of walls, items and traps. Following that, two experiments will be conducted towards both the influence of the action selection method as well as the influence of the rewards.

In order to show how these aspects influence the algorithm, certain elements will be generalised through all experiments. First of all, the learning rate and discount factor will be set to $\alpha = 0.1$ and $\gamma = 0.99$ for each experiment. The experiments are mainly set up to showcase how the exploration parameter influences the performance of the algorithm. Besides this, all experiments will be conducted on ten set seeds ranging from 0 to 9, meaning that they are replicable. The results, consisting of heat-maps of the environment and graphs showcasing the weighted average performance over these seeds, will be discussed in Section 5.

4.1 Small empty environment

This first experiment consists of a 6×6 environment, with $S = 36$. Here, the state space is equal to the size of the environment. Visible in Figure 4, the only present elements in this environment are the agent, the boundary walls and an exit. In order for the agent to learn properly, and for the results to highlight the exploration method, the reward of the end state will be set to +50. This experiment is intended to highlight how the algorithm performs in a very small environment and how this relates to the trade-off.

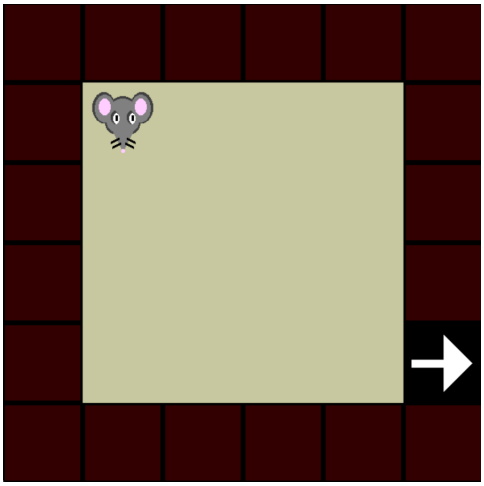
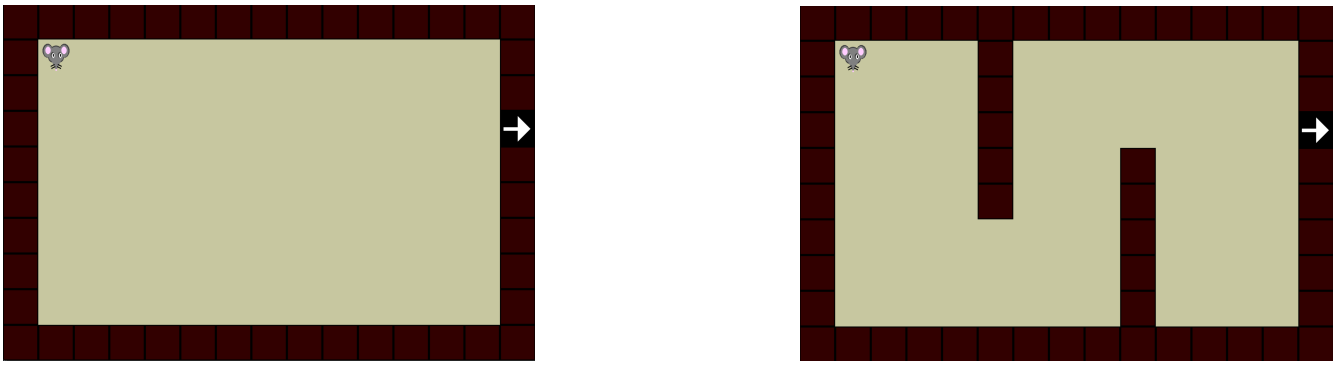


Figure 4: A small, empty environment, containing only the agent, walls and exit.

4.2 Influence of walls

In this experiment, the state dimensionality is increased and a comparison will be made as to how the addition of walls affects the algorithm's performance. Visible in Figure 5, we have two environments of 10×15 where one of the environments has two additional inner walls. Again, the spate space is equal to the size of the environment being $S = 150$ in both cases. The reward for the end state is set to $+50$. This experiment is intended to highlight how the algorithm performs in a bigger environment and how obstructions can affect its performance.



(a) No walls

(b) Containing inner Walls

Figure 5: Two larger environments, one of which contains additional walls.

4.3 Influence of items

The addition of items now drastically increases the state dimensionality in this experiment, visible in Figure 6. This dimensionality can be seen as a vector containing the length and width of the environment, alongside two binary entries for each item within the environment. When the agent picks up either one of the two cheeses, this changes the environment's states for all possible locations the agent can be in. This vector can be written as $S = [10, 15, 2, 2] = 600$ possible discrete states. Each cheese item returns a reward of $+15$. The end state has a reward of $+50$. This experiment is intended to highlight how sub-goals in the form of items affect how the algorithm performs.

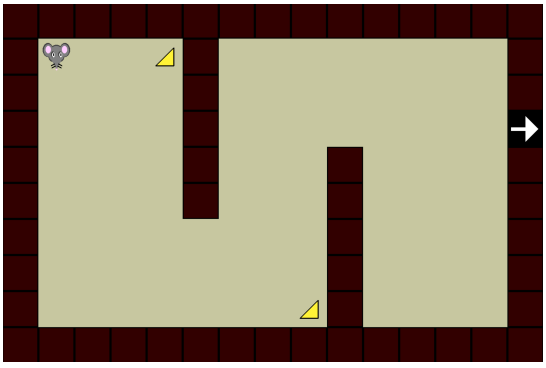


Figure 6: Environment containing two cheese items that can be picked up by the agent. This results in a positive reward.

4.4 Influence of traps: the full environment

This experiment showcases a full environment containing all possible elements, visible in Figure 7. The state space is even larger, with $S = [10, 15, 2, 2, 2] = 1200$ possible discrete states. Here, the cheese items grant a reward of $+15$, the cheese pile item has a reward of $+50$. The end state now has a reward of 0. This is done to stimulate the agent to pick up the cheese pile, as the reward of the traps is set to -100 and will reset the agent to its initial starting position, without starting a new episode.

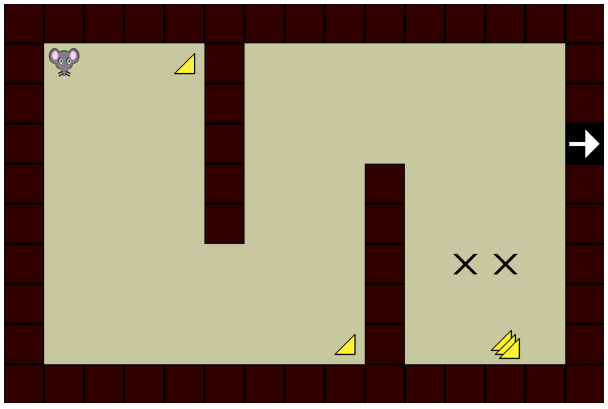


Figure 7: Environment containing two cheese items, traps and a cheese pile item which has a large positive reward. Traps result in a large negative reward and reset the agent to its starting point.

4.5 Influence of late rewards

The way in which feedback is granted to the agent through the environment is essential in its learning process. In order to visualise how this affects its performance, as well as how this influences the trade-off, this experiment emphasises the concept of late rewards. In this setting, the same environment of the experiment from 4.4 will be used, however, the agent will not receive an immediate reward upon entering a state with a cheese item. This reward will be granted when the agent has passed through such state when ultimately entering the end state. Because of this change in reward shaping, the rewards of the cheese items are set to $+50$ and the reward of the cheese pile is set to $+250$ in order for the agent to trace back their locations more effectively.

4.6 Upper-Confidence Bound

Besides ϵ-greedy, there is an abundance of action-selection methods applied within reinforcement learning algorithms. One of the more well-known methods is the Upper-Confidence Bound (UCB) method [SB18], which uses uncertainty in the action-value estimates for balancing between exploration and exploitation. In this experiment, the ϵ-greedy action-selection method will be replaced with the UCB method in order to show how this impacts the trade-off. This will be applied to the same environments as the experiments from 4.4 and 4.5. Formally, this method is defined as follows:

$$\pi_{\text{UCB}}(a) = \arg\max_a \left(Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}} \right)$$

5 Results

This section discusses the results from the experiments in Section 4. An overview is provided in Table 1 below. In the results, heat-maps are used to showcase the greedy paths of the agent. This is done by visualising their greedy action for each state. In situations where the state space is large because of present items, these heat-maps may specify a situation for when certain items are picked up. The heat-maps represent single runs from the experiments. This is because there is randomness involved that needs to be visualised. The randomness is part of the exploration technique used by the algorithm, which can cause certain states to have somewhat outlying greedy values. When this randomness is evened out over multiple runs, this key element of the algorithm would be visualised in a wrong manner when using heat-maps, since it then will not represent what is practically happening.

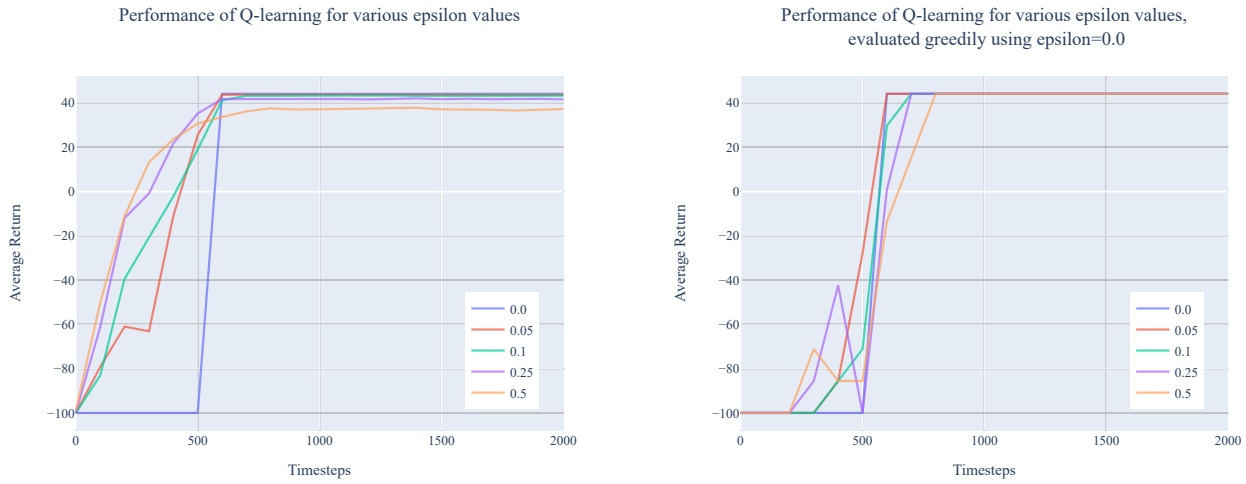
Alongside the heat-maps, graphs are used to visualise the performance of the algorithm. The shown graphs are weighted averages over ten runs using the seeds 0 to 9. The obtained curves are based on run-time evaluations of the algorithm. These evaluations occur roughly 40 times per run. During such evaluation, the algorithm will be placed in a duplicate of the environment with its current Q-table. Then, one episode with a maximum of 100 time-steps of the algorithm will be performed on this duplicate environment. The performance of the algorithm within this evaluation is based on the total obtained reward. This can either be done greedily, with $\epsilon = 0.0$, or with the actual ϵ value the algorithm uses to train with. Both options are valuable and will be used in this discussion.

Section	Experiment results	Pages
5.1	Small empty environment	15-16
5.2	Influence of walls	17-19
5.3	Influence of items	20-21
5.4	Influence of traps: the full environment	22-23
5.5	Influence of late rewards	24-25
5.6	Upper-Confidence Bound	26-28

Table 1: Overview of the experiment results discussed within this section.

5.1 Small empty environment

This section covers the results from the experiment in Section 4.1, featuring the small empty environment.



(a) Learning curves

(b) Greedy evaluation

Figure 8: Learning curves and greedy evaluation of Q-Learning for various values of ϵ, visualising the performance of the algorithm on a small empty environment. The curves depict the reward over time, averaged over ten runs with 2000 time-steps. Visible is that for all values the algorithm converges to the optimal solution at roughly 500 time-steps, indicating a similar performance.

(a) $\epsilon = 0.0$

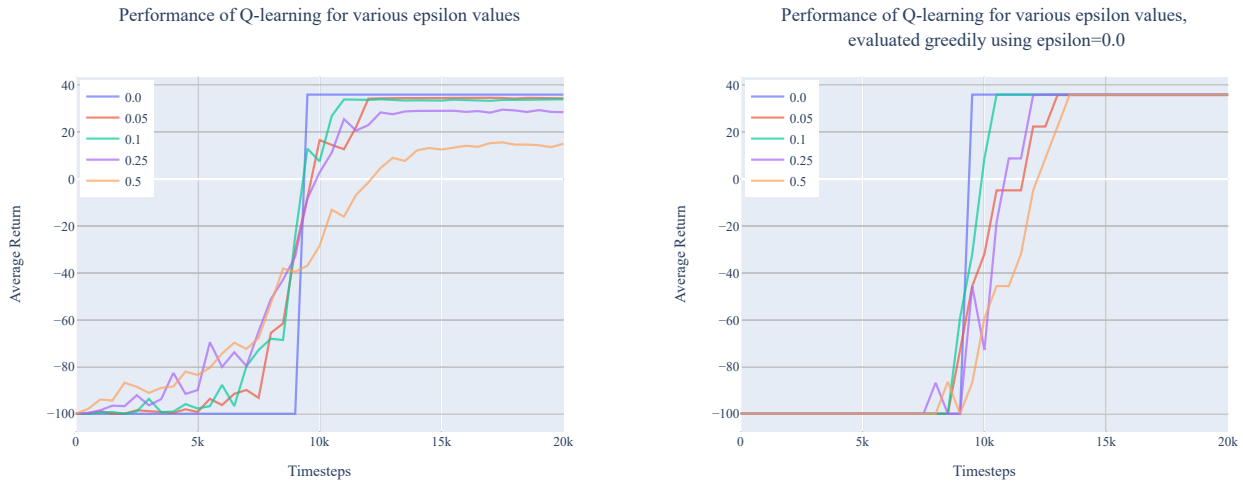
(b) $\epsilon = 0.5$

Figure 9: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.0$ in a and $\epsilon = 0.5$ in b in a small empty environment. Visible is that for $\epsilon = 0.0$ a clear distinctive path arises, whereas for $\epsilon = 0.5$ a broader area of the action-values related to the environment are high. This is due to the randomness involved, causing a wider area of the environment to be explored and filling the agents corresponding action-values with higher values.

The learning curves in Figure 8 indicate that for $\epsilon = 0.0$ the algorithm took longer to learn, but converges to the optimal solution the fastest alongside for $\epsilon = 0.1$, both having the best performance visible in the greedy evaluation. For $\epsilon = 0.5$, this took the longest, though it did find the optimal solution in acceptable time, indicating that the degree of exploration affects the performance only slightly. Important to notice is that the amount of time-steps is relatively low, meaning that the difference in performance for different ϵ values is marginal. When considering the heat-maps in Figure 9, we can see that there is a broader area covered by the algorithm under $\epsilon = 0.5$. Almost the whole set of greedy action-values corresponding to the environment locations contain high values, whereas for $\epsilon = 0.0$ this is not the case. This is due to the level of exploration used by the algorithm, which for $\epsilon = 0.5$ causes a random action 50% of the time, causing the whole environment to be covered with high action values. In this case, the results show that the level of exploration is not as important, indicating that in a simple, empty environment the trade-off is not necessarily present.

5.2 Influence of walls

This section covers the results from the experiment in Section 4.2, featuring a comparison between two larger environments, with one of which containing additional inner walls.



(a) Learning curves

(b) Greedy evaluation

Figure 10: Learning curves and greedy evaluation of Q-Learning for various values of ϵ, visualising the performance of the algorithm on a larger environment without walls. The curves depict the reward over time, averaged over ten runs with 20k time-steps. Visible is that for $\epsilon = 0.0$ the algorithm performed best, converging to the optimal solution at roughly 9k time-steps. The learning curves show that the higher the ϵ value, the more gradually the algorithm learns. The algorithm converges to the optimal solution for all ϵ values, which can be seen in the greedy evaluation. Note that for $\epsilon = 0.1$, the algorithm performed better than all other values, except $\epsilon = 0.0$.

(a) $\epsilon = 0.0$

(b) $\epsilon = 0.1$

Figure 11: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.0$ in a and $\epsilon = 0.1$ in b in a larger environment without inner walls. Visible in both heat-maps is a clear path to the exit, where for $\epsilon = 0.1$ this path is slightly wider.

(a) Learning curves

(b) Greedy evaluation

Figure 12: Learning curves and greedy evaluation of Q-Learning for various values of ϵ, visualising the performance of the algorithm on a larger environment containing inner walls. The curves depict the reward over time, averaged over ten runs with 30k time-steps. Visible is that for most values the algorithm has steep learning curves around 15k time-steps, where for $\epsilon = 0.0$ this arises around 18k time-steps and for $\epsilon = 0.5$ this is more graduate. For $\epsilon = 0.1$ the algorithm performs best, finding the optimal solution the fastest, just before $\epsilon = 0.05$ and $\epsilon = 0.25$. Only for $\epsilon = 0.0$, the optimal solution has not been found, as seen in the greedy evaluation. This indicates that the addition of inner walls demands a degree of exploration.

(a) $\epsilon = 0.0$

(b) $\epsilon = 0.1$

Figure 13: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.0$ in a and $\epsilon = 0.1$ in b in a larger environment containing inner walls. Visible is that for $\epsilon = 0.0$, there is a distinctive path towards the exit. For $\epsilon = 0.1$, there has been more exploration, resulting in a wider path with a broader amount of high action-values.

When comparing the graphs in Figures 10 and 12, it becomes apparent that the addition of walls in this larger environment demands exploration. Whereas for the environment without walls the algorithm performed best under $\epsilon = 0.0$, it did not converge to the optimal solution under this value in the environment with inner walls. This shows that exploration becomes important when the environment becomes more complex. Important to notice is that for the environment with walls, the algorithm took roughly 5000 time-steps longer to converge. Though the state dimensions are the same, the addition of walls made the environment more complex.

In the environment without inner walls, the algorithm converges to the optimal solution for all values for ϵ, with $\epsilon = 0.0$ having the best performance followed by $\epsilon = 0.1$. It is worth noting that for $\epsilon = 0.05$ the algorithm performed worse than those two. At first glance this seems abnormal, as the expectation would be that there would be a 'sweet spot' for the degree of exploration around which the other values would centre. In this case, this might be caused by the way the rewards of the environment are shaped. For $\epsilon = 0.0$, there is still some degree of exploration since each step results in a reward of -1. When this accumulates over time, the agent is forced to take a greedy action towards a state that has not been explored yet, since its action-value will be higher as a result of exploiting the other states with a negative reward. In this case, the algorithm might be guided in some sense, resulting in a somewhat distorted perception of the performance for $\epsilon = 0.0$. This means the algorithm might actually explore more than for $\epsilon = 0.05$ in a guided manner, although this would only occur at the beginning of a run. Visible in the heat-maps in Figure 11 is a clear path to the exit for $\epsilon = 0.0$ and $\epsilon = 0.1$, where it can be seen that for the latter the algorithm has explored more neighbouring states due to the involvement of random actions.

For the environment with walls, the algorithm did not find an optimal solution under $\epsilon = 0.0$. All other values for ϵ did result in convergence to the optimal solution, indicating that the addition of walls impacted the algorithm in such a way that there was a demand for exploration. Here, the trade-off becomes apparent, since for $\epsilon = 0.1$ the algorithm performed best, followed by $\epsilon = 0.05$ and $\epsilon = 0.25$. Visible in the heat-maps in Figure 13, a clear path towards the exit is visualised again for both shown ϵ values. Exploration under $\epsilon = 0.1$ resulted in a better performance, indicating that the path shown for $\epsilon = 0.0$ is not the optimal path.

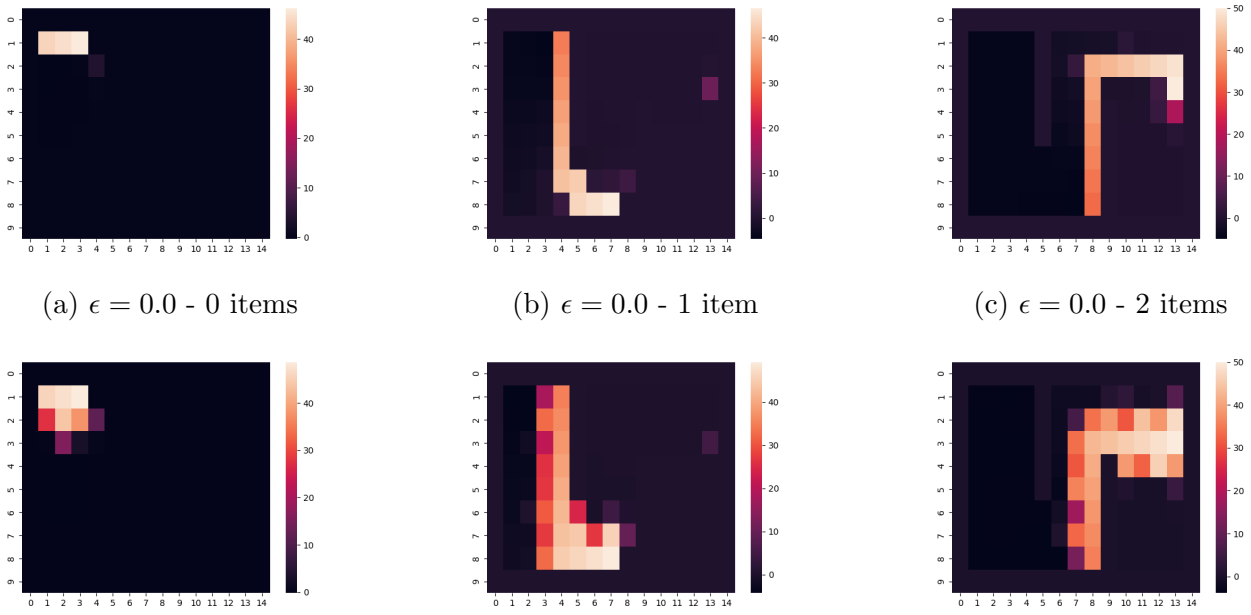
5.3 Influence of items

This section covers the results from the experiment in Section 4.3, featuring an environment with two cheese items.

(a) Learning curves

(b) Greedy evaluation

Figure 14: Learning curves and greedy evaluation of Q-Learning for various values of ϵ, visualising the performance of the algorithm on an environment containing two cheese items. The curves depict the reward over time, averaged over ten runs with 30k time-steps. Most curves start to converge around 15k time-steps, apart from $\epsilon = 0.5$ which does this more gradually. The algorithm performs best for $\epsilon = 0.1$, converging to the optimal solution the fastest, followed by $\epsilon = 0.25$ and $\epsilon = 0.05$. For $\epsilon = 0.0$, the algorithm does not converge to the optimal solution, visible in the greedy evaluation. This indicates that additional items within the environment demand for a degree of exploration.



(a) $\epsilon = 0.0$ - 0 items (b) $\epsilon = 0.0$ - 1 item (c) $\epsilon = 0.0$ - 2 items

(d) $\epsilon = 0.1$ - 0 items (e) $\epsilon = 0.1$ - 1 item (f) $\epsilon = 0.1$ - 2 items

Figure 15: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.0$ in *a,b,c* and $\epsilon = 0.1$ in *d,e,f*. The heat-maps represent three different situations, based on the order of picked up items. This results in a path, which traces the item locations towards the exit. Visible is a clear path of greedy action-values for $\epsilon = 0.0$, as well as a broader, more explored path for $\epsilon = 0.1$

In Figure 14, it can be seen that for $\epsilon = 0.1$ the algorithm performs best, converging to the optimal solution the fastest. Other values such as $\epsilon = 0.25$ and $\epsilon = 0.05$ resulted in a good performance as well. The algorithm converged to the optimal solution for all ϵ values, except for $\epsilon = 0.0$, indicating that the addition of items as a form of sub-goals within the environment demands a degree of exploration. Visible is that all curves have a slight peak around 5000 time-steps, likely corresponding to the convergence of the action-values in regard to finding the first cheese item. This can be seen in both graphs for learning curves as well as for the greedy evaluation. The heat-maps in Figure 15 show that for $\epsilon = 0.1$ the algorithm explored a broader area around the optimal path. The path visualised for $\epsilon = 0.0$ is clear, but not the optimal path. This shows that the exploration used by the algorithm resulted in finding a better path, instead of exploiting a sub-optimal solution.

5.4 Influence of traps: the full environment

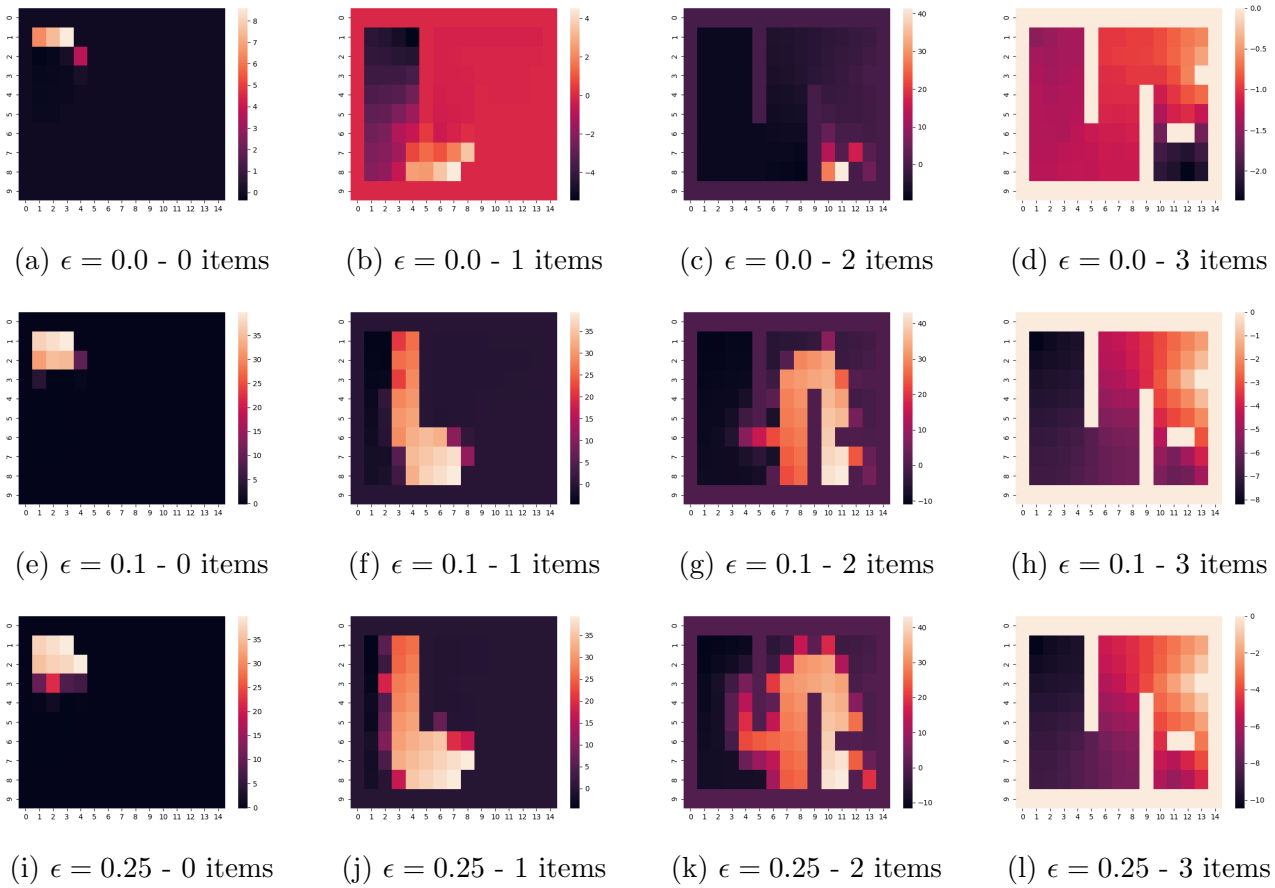
This section covers the results from the experiment in Section 4.4, featuring a full environment containing cheese and cheese pile items, as well as traps.

(a) Learning curves

(b) Greedy evaluation

Figure 16: Learning curves and greedy evaluation of Q-Learning for various values of ϵ, visualising the performance of the algorithm on an environment containing all elements. The curves depict the reward over time, averaged over ten runs with 100k time-steps. Most noticeable is that for $\epsilon = 0.0$ the algorithm performs the worst, converging into a solution with a local optimum. For all other values, the learning curves are very graduate and the algorithm converges to the optimal solution, with $\epsilon = 0.1$ showing the best performance followed by $\epsilon = 0.25$.

In Figure 16, it is most noticeable that for $\epsilon = 0.0$ the algorithm performed worst. The algorithm did not find an optimal solution, which is also visualised in the heat-maps in Figure 16. At 40k time-steps, a dip in performance can be seen. This dip should normally be evened out over ten runs, however, this is not the case for $\epsilon = 0.0$ given the way that the algorithm was set up. When run over ten seeds, the algorithm would perform the exact same behaviour regardless of the set seed. This is most likely due to the fact that it makes use of the numpy argmax functionality. In the case of $\epsilon = 0.0$, given situations where there are multiple max arguments to choose from, this function will always return the first action, explaining why repeating behaviour may occur for each seed. This could be overcome by implementing a functionality that chooses a random action out of all max arguments, however, this could potentially influence the exploration of the algorithm as well.



(a) $\epsilon = 0.0$ - 0 items (b) $\epsilon = 0.0$ - 1 items (c) $\epsilon = 0.0$ - 2 items (d) $\epsilon = 0.0$ - 3 items

(e) $\epsilon = 0.1$ - 0 items (f) $\epsilon = 0.1$ - 1 items (g) $\epsilon = 0.1$ - 2 items (h) $\epsilon = 0.1$ - 3 items

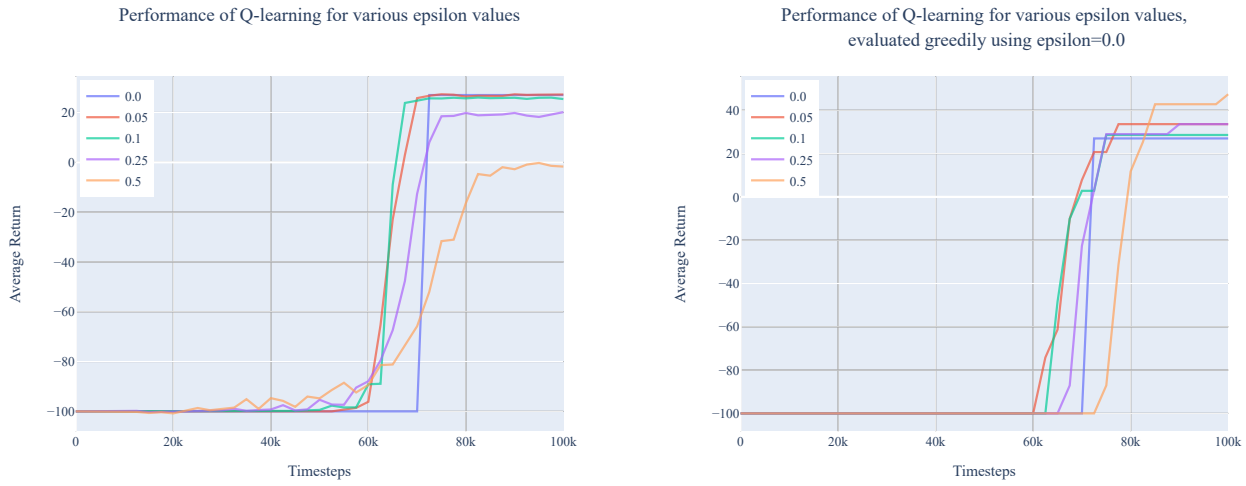
(i) $\epsilon = 0.25$ - 0 items (j) $\epsilon = 0.25$ - 1 items (k) $\epsilon = 0.25$ - 2 items (l) $\epsilon = 0.25$ - 3 items

Figure 17: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.0$ in *a,b,c,d*, $\epsilon = 0.1$ in *e,f,g,h* and $\epsilon = 0.25$ in *i,j,k,l*. The heat-maps represent four different situations, based on the order of picked up items. Most apparent in *b* and *c* for $\epsilon = 0.0$ there is no traceable path to be seen, indicating a bad performance. The values represented in the heat-maps for this value are lower than for $\epsilon = 0.1$ and $\epsilon = 0.25$, which do indicate distinctive paths. Here, the visualised action-values show a path between the three items which avoids the traps. The final heat-maps show inverted colours. This is due to the fact that there is no obtainable positive reward left at this point. Instead, a path with the least negative reward towards the exit is visible.

Further visible in Figure 16 is that for all other values of ϵ, the algorithm converged to the optimal solution. For $\epsilon = 0.5$ a huge dip can be seen after 50k time-steps. This could potentially be explained by the fact that, at that point, it had learned the optimal path towards the cheese pile item. After finding this path, exploration causes the agent to walk in to the nearby traps, resulting in a reward of -100 and having to start over. The heat-maps in Figure 17 further visualise that for $\epsilon = 0.1$ and $\epsilon = 0.25$ an optimal path was found, including broad areas of explored states surrounding this path. Given the graphs, this area of explored states was required by the algorithm in order to learn the optimal path most efficiently. These results emphasise that the trade-off is prevalent in a complex environment.

5.5 Influence of late rewards

This section covers the results from the experiment in Section 4.5, featuring the full environment with delayed feedback of the rewards.

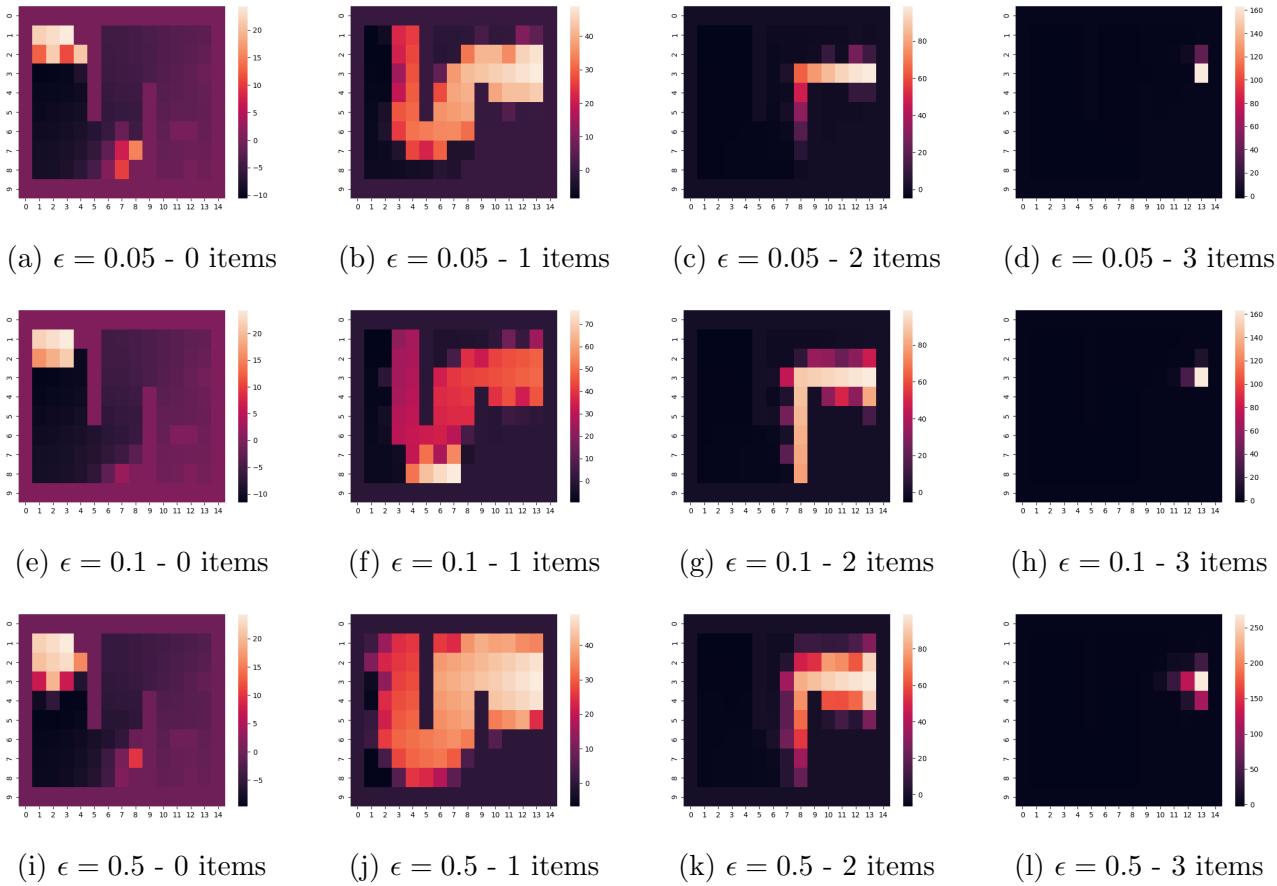


(a) Learning curves

(b) Greedy evaluation

Figure 18: Learning curves and greedy evaluation of Q-learning for various values of ϵ, visualising the performance of the algorithm on an environment with late rewards. The curves depict the reward over time, averaged over ten runs with 100k time-steps. Visible is that for all values, the learning curves start to converge after roughly 65k time-steps. This is highly likely caused by the late rewards, for which the agent needs a significant amount of time to correctly learn from. None of the ϵ values allow for a convergence to the optimal solution. Despite converging latest, $\epsilon = 0.5$ shows the best performance, reaching an average reward higher than all other values performances. This indicates that this change in reward shaping demands a lot of exploration from the algorithm.

Visible in Figure 18 is that for all values of ϵ the algorithm learns slow. This could be explained by the fact that the positive rewards of the items are granted to the agent upon arriving in the end state. The agent would need a significant amount of time to correctly learn to trace back this reward to the item locations. None of the ϵ values resulted in an optimal solution. Visible on the y-axis, the indicated average rewards are fairly low considering the generous way rewards are set up within this experiment. For $\epsilon = 0.5$, the algorithm shows the best performance, increasing the average return steadily at a higher rate than the other ϵ values. When considering the heat-maps in Figure 19, it can be seen that the algorithm has not yet traced back the cheese pile item location yet. Combining both these findings, it can be assumed that the algorithm did not have enough time to converge fully to the optimal solution. What is important to notice is that only this change in how feedback is granted to the agent showed a great difference in how various ϵ values affected the performance of the algorithm. These results emphasise that this affects the trade-off, showing that a high exploration parameter positively affects the performance in this case.



(a) $\epsilon = 0.05$ - 0 items (b) $\epsilon = 0.05$ - 1 items (c) $\epsilon = 0.05$ - 2 items (d) $\epsilon = 0.05$ - 3 items

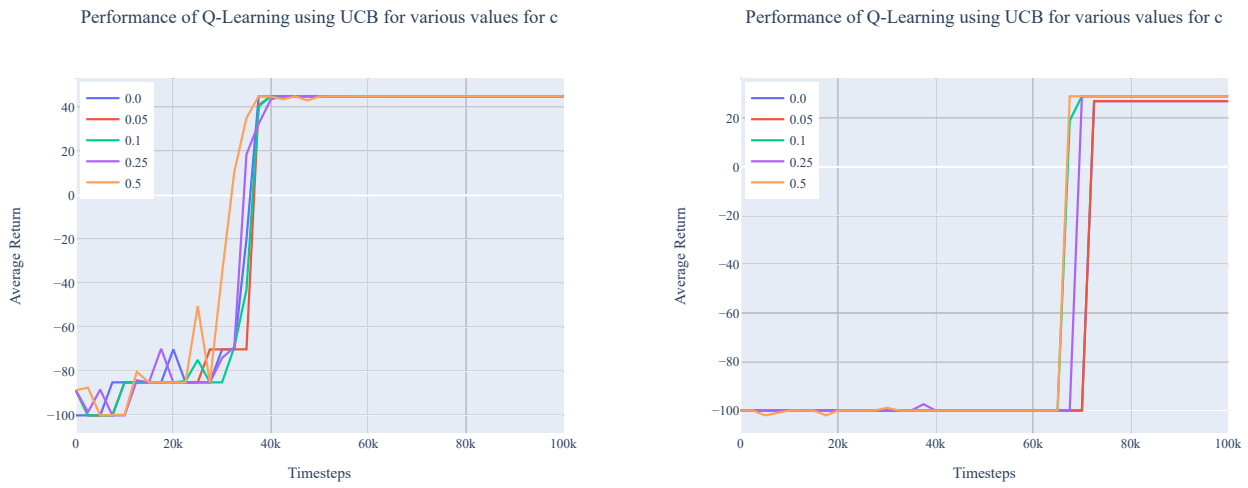
(e) $\epsilon = 0.1$ - 0 items (f) $\epsilon = 0.1$ - 1 items (g) $\epsilon = 0.1$ - 2 items (h) $\epsilon = 0.1$ - 3 items

(i) $\epsilon = 0.5$ - 0 items (j) $\epsilon = 0.5$ - 1 items (k) $\epsilon = 0.5$ - 2 items (l) $\epsilon = 0.5$ - 3 items

Figure 19: Heat-maps visualising greedy action-values of the algorithm for $\epsilon = 0.05$ in *a,b,c,d*, $\epsilon = 0.1$ in *e,f,g,h* and $\epsilon = 0.5$ in *i,j,k,l*. The heat-maps represent four different situations, based on the order of picked up items. Here, a clear path can be traced, however for all values of ϵ, it avoids the cheese pile item. The final heat-maps show inverted colours. This is due to the fact that there is no obtainable positive reward left at this point. Instead, a path with the least negative reward towards the exit is visible.

5.6 Upper-Confidence Bound

This section covers the results from the experiment in Section 4.6, featuring the same environments as in 4.4 and 4.5. Here, the ϵ-greedy action-selection policy is replaced with the UCB method.

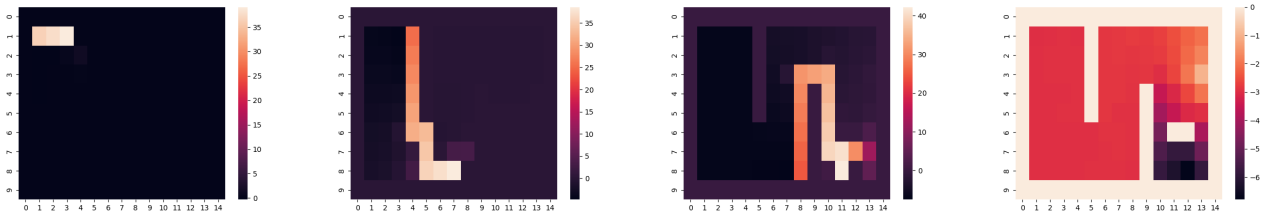


(a) Full environment

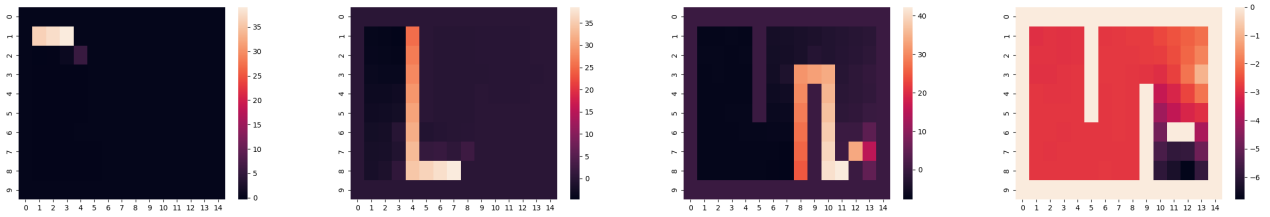
(b) Late rewards

Figure 20: Learning curves of Q-Learning using a UCB policy for various values of c. The performance of the algorithm is visualised on the environment containing all elements from experiment 4.4, as well as an environment with late rewards from experiment 4.5. The curves depict the reward over time, averaged over ten runs with 100k time-steps. Visible is that under this policy, the learning curves show very similar performance for various exploration values. The algorithm shows the best performance under $c = 0.5$ on both environments, though it is marginal. In the environment with late rewards, the algorithm does not converge to the optimal solution.

The results of this experiment show that when using a different policy, this drastically changes the algorithms learning process. When comparing the results from Figure 20a to those of Figure 16, it becomes clear that the algorithm significantly performs better under a UCB policy, compared to ϵ-greedy. The algorithm reaches the optimal solution relatively fast, having steep curves that are all very similar. This may indicate that under this policy, the trade-off is dealt with in a more concise manner. Initially, UCB explores more to systematically reduce uncertainty, after which its exploration reduces over time. This could benefit the algorithm when dealing with very complex environments, where exploration is needed in gradations. In 20b, it is visible that the algorithm still does not converge to the optimal solution. When compared to Figure 18, it seems to perform similar and may need more time to converge to the optimal solution. The heat-maps in Figures 21 and 22 show the greedy action-values for both environments. Here, it can clearly be seen that for $c = 0.0$ and $c = 0.5$ the algorithm shows very similar performance on both environments. In Figure 21, a clear path is seen that passes through all three items and avoids traps, after which it leads to the exit. In Figure 22, this path still avoids the cheese pile item in the environment with late rewards. The greedy path has not been traced back towards the second and third items, indicating it may still need time to converge to this solution.

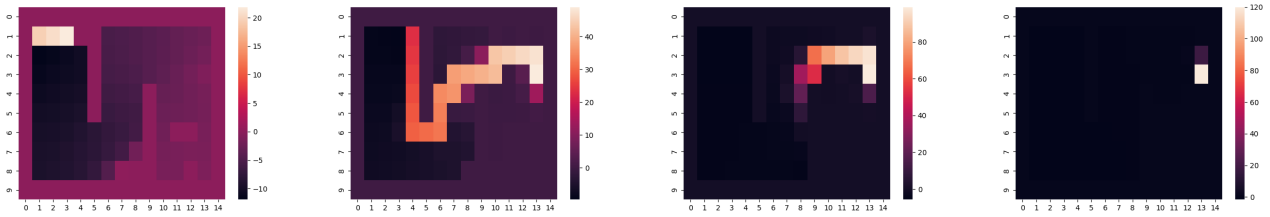


(a) $c = 0.0$ - 0 items (b) $c = 0.0$ - 1 items (c) $c = 0.0$ - 2 items (d) $c = 0.0$ - 3 items

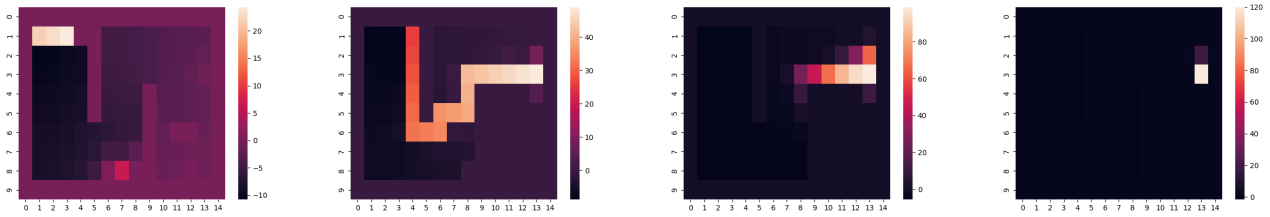


(e) $c = 0.5$ - 0 items (f) $c = 0.5$ - 1 items (g) $c = 0.5$ - 2 items (h) $c = 0.5$ - 3 items

Figure 21: Heat-maps visualising greedy action-values of the algorithm for $c = 0.0$ in *a,b,c,d*, and $c = 0.5$ in *e,f,g,h*. The heat-maps represent four different situations, based on the order of picked up items. The final heat-maps show inverted colours. This is due to the fact that there is no obtainable positive reward left at this point. Instead, a path with the least negative reward towards the exit is visible. For both values, the algorithm shows a similar greedy path, which passes through all three items and avoids the traps.



(a) $c = 0.0$ - 0 items (b) $c = 0.0$ - 1 items (c) $c = 0.0$ - 2 items (d) $c = 0.0$ - 3 items



(e) $c = 0.5$ - 0 items (f) $c = 0.5$ - 1 items (g) $c = 0.5$ - 2 items (h) $c = 0.5$ - 3 items

Figure 22: Heat-maps visualising greedy action-values of the algorithm for $c = 0.0$ in *a,b,c,d*, and $c = 0.5$ in *e,f,g,h*. The heat-maps represent four different situations, based on the order of picked up items. The final heat-maps show inverted colours. This is due to the fact that there is no obtainable positive reward left at this point. Instead, a path with the least negative reward towards the exit is visible. For both values, the algorithm shows a similar greedy path, which does not pass through the cheese pile item.

6 Conclusions

Based on the results from the experiments, multiple conclusions can be made. First of all, the dilemma between exploration and exploitation is less prevalent in the case of a small, simple environment. The results from experiment 4.1 show that there is a minimal difference in performance between various values for the ϵ parameter, indicating that in such case there is no real trade-off existent between exploration and exploitation. For $\epsilon = 0.0$ and $\epsilon = 0.5$, the algorithm converged to the optimal solution, only taking the slightest amount of time longer for the latter. In such cases, it is not of importance how much the algorithm makes use of exploration, since it will converge to the optimal solution nevertheless.

In situations where the environment is more complex, such as an increase of the environment's size, or the addition of sub-goals in the form of items, the trade-off does become prevalent. The addition of inner walls in experiment 4.2 showed that this already demanded a degree of exploration in the algorithm in order to have a better performance. This only increased with the introduction of additional elements in experiments 4.3 and 4.4. In the cases of more complex environments, the algorithm could not converge to an optimal solution when making use of $\epsilon = 0.0$. This implies that exploration, at least to some degree, is needed for an algorithm like Q-Learning to properly learn the correct behaviour. When an environment demands exploration, $\epsilon = 0.0$ will never be optimal.

Now, the real dilemma of this trade-off comes forth in situations where it is hard to pre-determine what degree of exploration would be most suitable. The experiment results showed that for various situations, the algorithm performed best under different ϵ values. The same can be said for the exploration parameter used in the UCB policy from experiment 4.6, though it significantly outperformed the ϵ-greedy policy. This leads to the conclusion that this dilemma is very important in cases of a more complex environment, and that there is always a trade-off between what degree of exploration works best. Given different situations, the degree of exploration is always dependent on the environment and its complexity, making it hard to determine what degree of exploration would lead to the optimal behaviour beforehand. This should always be taken into account when faced with a reinforcement learning problem.

Alongside these conclusions on the degree of exploration, there are multiple elements within reinforcement learning that influence this trade-off. What came forth during the design of the environments was that each subtle change to the environment led to completely different outcomes. The performance of the algorithm was influenced in such a manner, that even slight alterations of elements within the environment, such as moving the exit down one location, influenced the overall performance of the algorithm. Besides the complexity and shape of the environment, the way in which the agent gets feedback from the rewards has to be taken into account as well. In experiment 4.5, this is highlighted in the form of late rewards, but it goes much deeper than that. The negative reward of each step influences the exploration of the algorithm as well. When using an exploration parameter such as $\epsilon = 0.0$, this does not mean that the algorithm will not explore at all. The Q-Learning algorithm will be slightly guided towards a more positive reward state, since the cumulative addition of a negative reward will force the agent to take a greedy action towards states that have not yet been explored.

6.1 Answers to the Research Questions

What is the importance of the exploration-exploitation trade-off in reinforcement learning?

The importance of the trade-off is that it is present in a variety of problems and influenced by multiple factors. The environment, the used algorithm and the way the agent gets feedback through rewards all affect the trade-off in different ways. Each problem requires a different degree of exploration, which needs to be balanced with exploitation. Its balance is essential for the performance of the algorithm, while also being very dependent on the situation, resulting in a trade-off that needs to be considered in almost all cases.

What is the best way to implement and showcase the effects of this trade-off in EduGym?

The best way to showcase the effects of the trade-off is by showcasing different situations that affect the trade-off in their own way. This thesis aimed to show that by demonstrating variations of an environment that grows in complexity, the algorithm was influenced differently in each case, showing that the trade-off had to be considered at all times. An implementation of a straightforward and interactive environment, like the Mouse Maze environment implemented for this thesis, would allow for the user to focus on the underlying problem. By show-casing multiple variations of this environment, the user could be demonstrated how different situations affect the trade-off differently, giving a broad idea of how this trade-off could affect more complex reinforcement learning problems. Alongside this, the shown graphs and heat-maps would greatly benefit the user in its understanding of how different exploration parameters influence the agent's behaviour.

6.2 Further Research

The goal of this thesis was to provide an assessment on the importance of the exploration-exploitation trade-off, as well as providing a constructive approach as to how this could be implemented for EduGym. The delivered implementation consists of multiple variations of an environment, all meant to feature certain aspects that could influence this trade-off. This idea of featuring how different situations can affect the trade-off differently is useful, but could still be improved. Numerous elements affect the trade-off in different ways, such as the way the agent receives feedback as well as how the actual policies are programmed. Factors such as a negative reward of -1 for each step, or a utilised numpy argmax function in the action selection functionality should be taken into consideration for further research. Another thing that could be further researched for this implementation, is the addition of an interactive notebook. For now, this thesis only delivered the implementation of the Mouse Maze environment in Python code, whereas EduGym makes use of such notebooks for each of the implementations.

References

[Aue02] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, OpenAI Gym, 2016. arXiv: 1606.01540.

[CL20] Jesse Clifton and Eric Laber. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7:279–301, 2020.

[Fru19] Ronan Fruit. *Exploration-exploitation dilemma in Reinforcement Learning under various form of prior knowledge*. PhD thesis, Université de Lille, Lille, 2019.

[GR13] Frédérick Garcia and Emmanuel Rachelson. *Markov Decision Processes in Artificial Intelligence*, chapter 1: Markov Decision Processes, pages 1–38. John Wiley & Sons, Ltd, 2013. doi: 10.1002/9781118557426.

[KLM95] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. An introduction to reinforcement learning. In *The Biology and Technology of Intelligent Autonomous Agents*, pages 90–127, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[MMBY+23a] Thomas M. Moerland, Matthias Müller-Brockhausen, Zhao Yang, Andrius Bernatavicius, Koen Ponse, Tom Kouwenhoven, Andreas Sauter, Michiel van der Meer, Bram Renting, and Aske Plaat. Edugym: an environment and notebook suite for reinforcement learning education, 2023. preprint arXiv: 2311.10590.

[MMBY+23b] Thomas M. Moerland, Matthias Müller-Brockhausen, Zhao Yang, Andrius Bernatavicius, Koen Ponse, Tom Kouwenhoven, Andreas Sauter, Michiel van der Meer, Bram Renting, and Aske Plaat. Edugym website, 2023. URL: https://sites.google.com/view/edu-gym. Accessed: February - July 2024.

[OGNJ17] Yi Ouyang, Mukul Gagrani, Ashutosh Nayyar, and Rahul Jain. Learning unknown markov decision processes: A thompson sampling approach. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[RRK+17] Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *CoRR*, 2017. arXiv: 1707.02038.

[SB99] Richard S. Sutton and Andrew G. Barto. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Massachusetts, second edition, 2018.

[Tho33] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[TTK+24] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shenand, and Omar G. Younis. Gymnasium, 2024. URL: https://zenodo.org/record/8127025.

[VBW15] Sofia Villar, Jack Bowden, and James Wason. Multi-armed bandit models for the optimal design of clinical trials: Benefits and challenges. *Statistical Science*, 30:199–215, 2015.

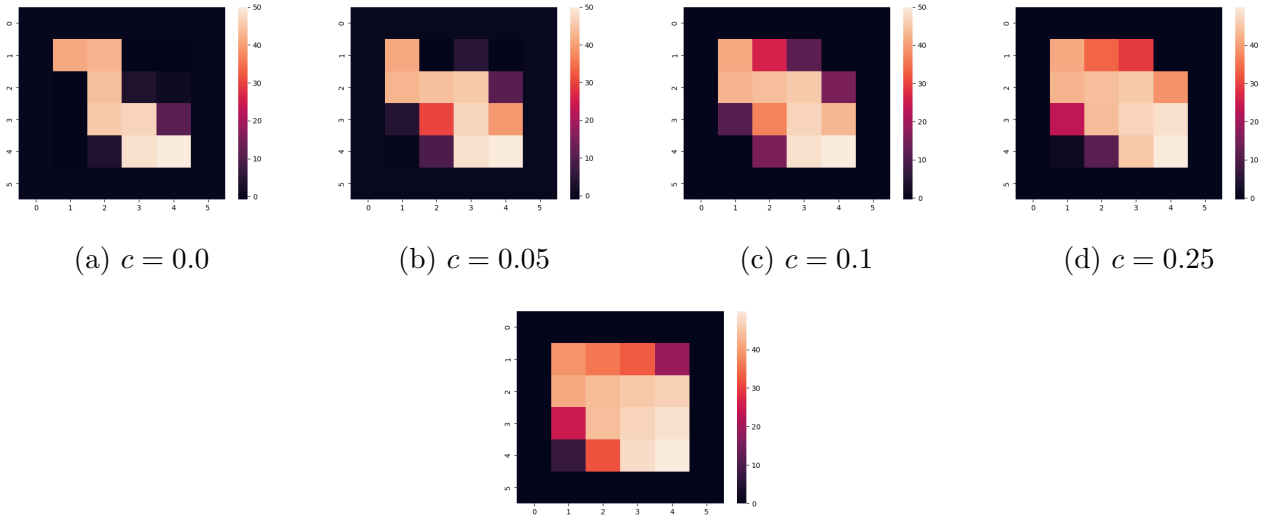
[Wat89] Chris J. C. H. Watkins. *Learning from Delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[WGL17] Huasen Wu, Xueying Guo, and Xin Liu. Adaptive exploration-exploitation tradeoff for opportunistic bandits. *CoRR*, 2017. arXiv: 1709.04004.

A Experiment heat-maps

This appendix features all heat-maps obtained from the experiments.

Experiment 4.1: Small empty environment

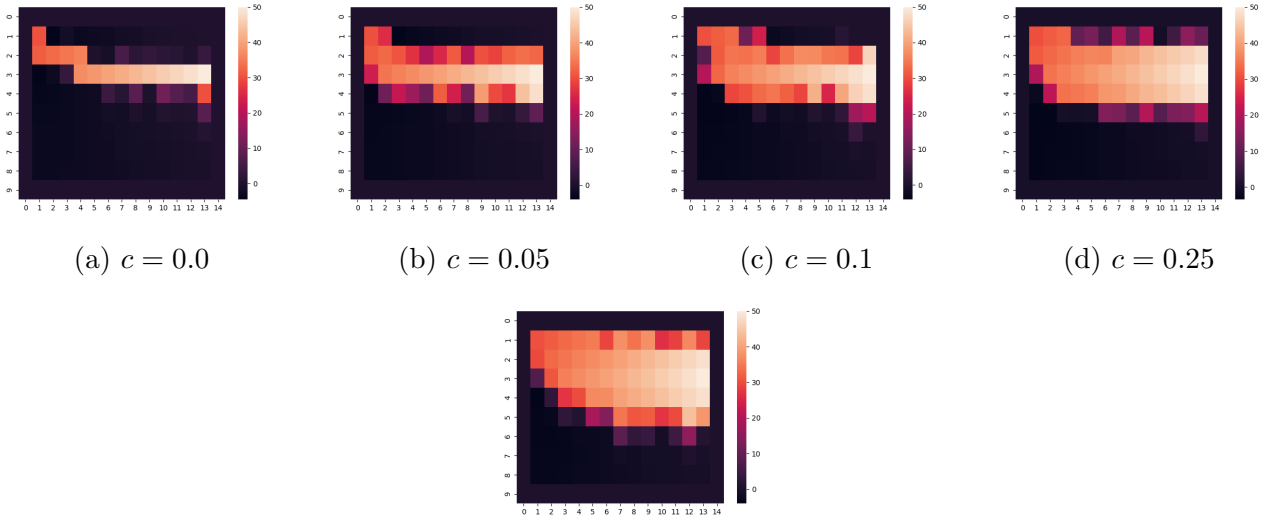


(a) $c = 0.0$ (b) $c = 0.05$ (c) $c = 0.1$ (d) $c = 0.25$

(e) $c = 0.5$

Figure 23: heat-maps experiment 4.1

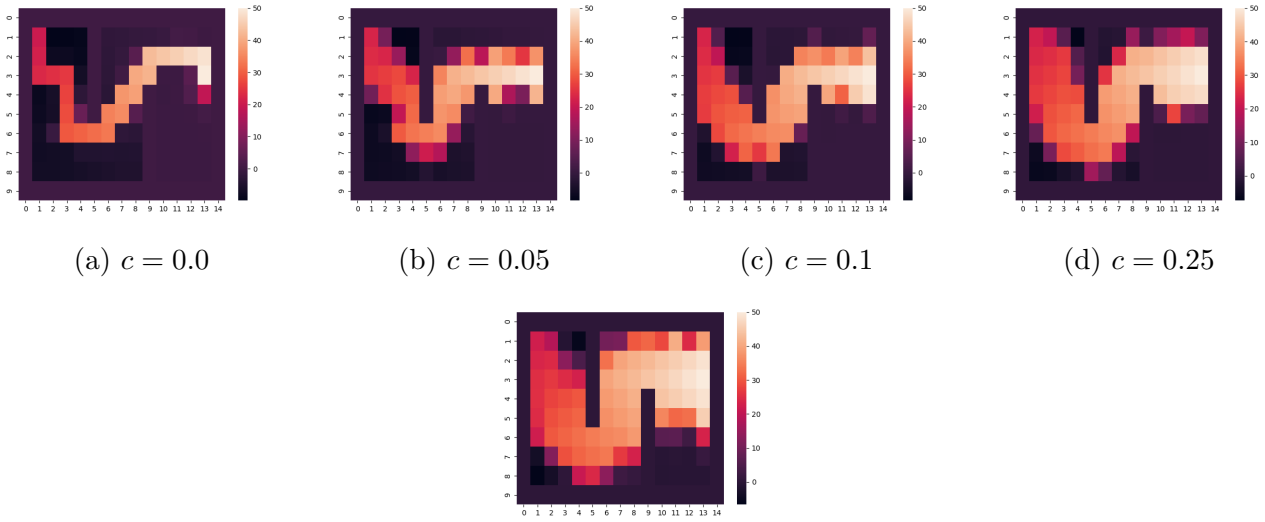
Experiment 4.2: Influence of walls



(a) $c = 0.0$ (b) $c = 0.05$ (c) $c = 0.1$ (d) $c = 0.25$

(e) $c = 0.5$

Figure 24: heat-maps experiment 4.2: no walls

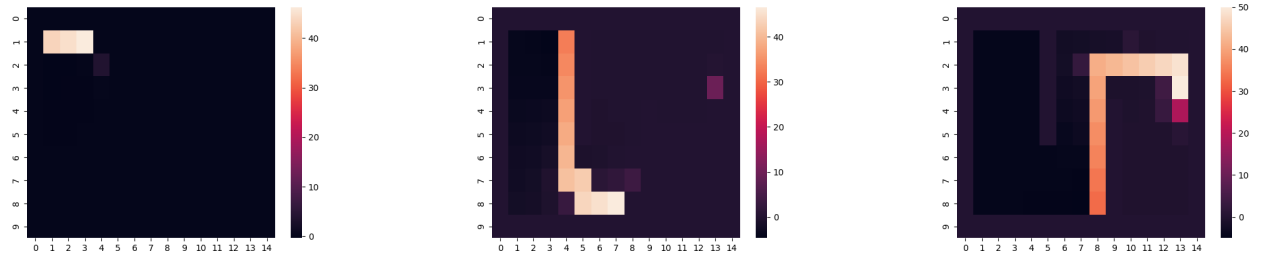


(a) $c = 0.0$ (b) $c = 0.05$ (c) $c = 0.1$ (d) $c = 0.25$

(e) $c = 0.5$

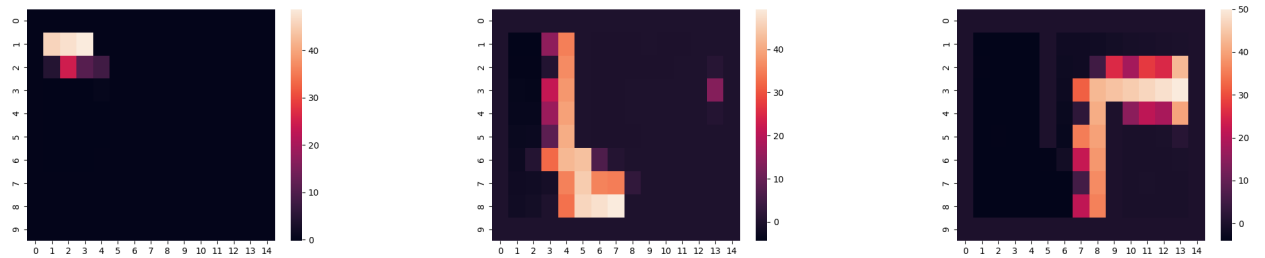
Figure 25: heat-maps experiment 4.2: walls

Experiment 4.3: Influence of Items
Cxx indicates the (order of) picked up items for each situation.



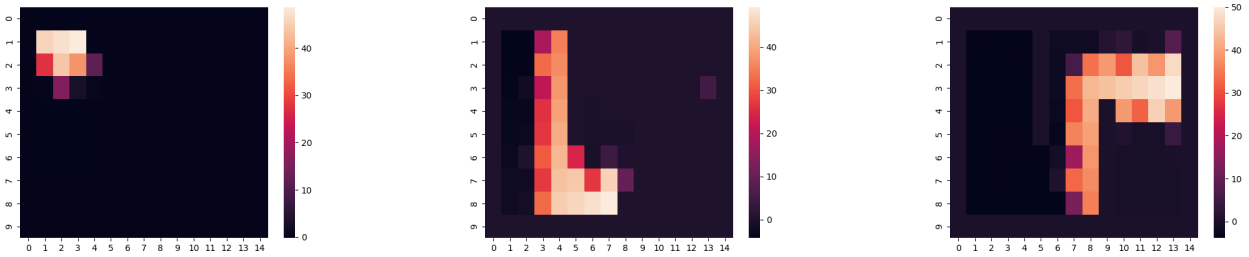
(a) C00 (b) C10 (c) C11

Figure 26: heat-maps experiment 4.3 for $\epsilon = 0.0$.



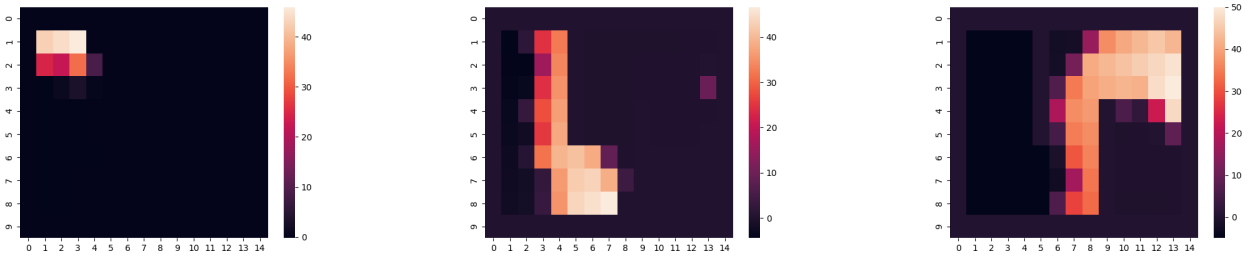
(a) C00 (b) C10 (c) C11

Figure 27: heat-maps experiment 4.3 for $\epsilon = 0.05$.



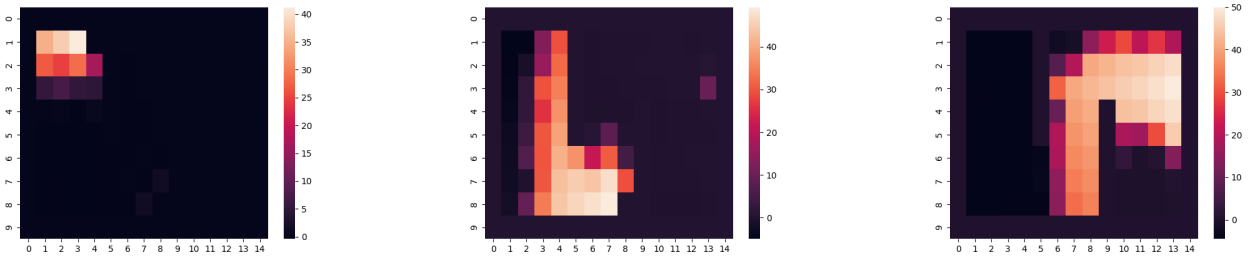
(a) C00 (b) C10 (c) C11

Figure 28: heat-maps experiment 4.3 for $\epsilon = 0.1$.



(a) C00 (b) C10 (c) C11

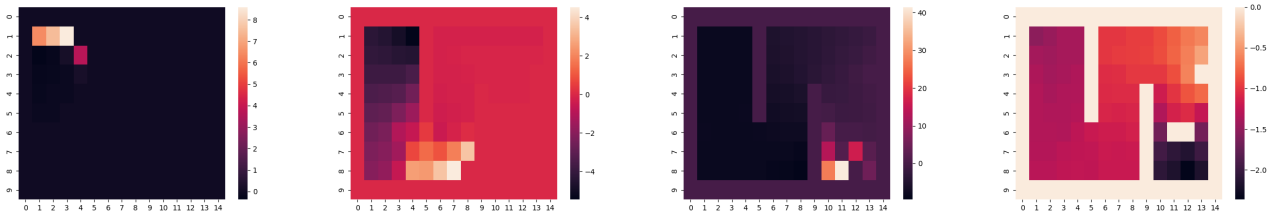
Figure 29: heat-maps experiment 4.3 for $\epsilon = 0.25$.



(a) C00 (b) C10 (c) C11

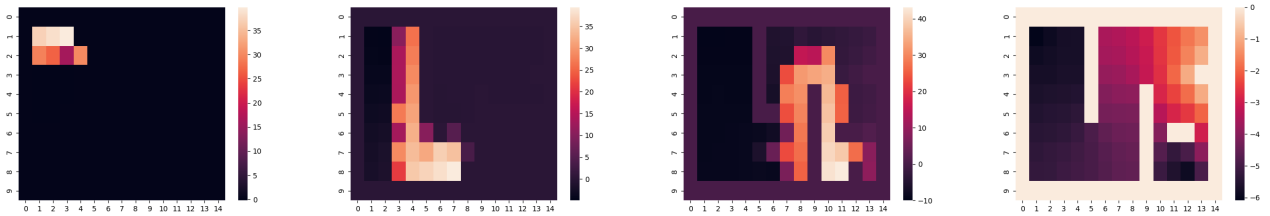
Figure 30: heat-maps experiment 4.3 for $\epsilon = 0.5$.

Experiment 4.4: Influence of traps: the full environment
Cxxx indicates the (order of) picked up items for each situation.



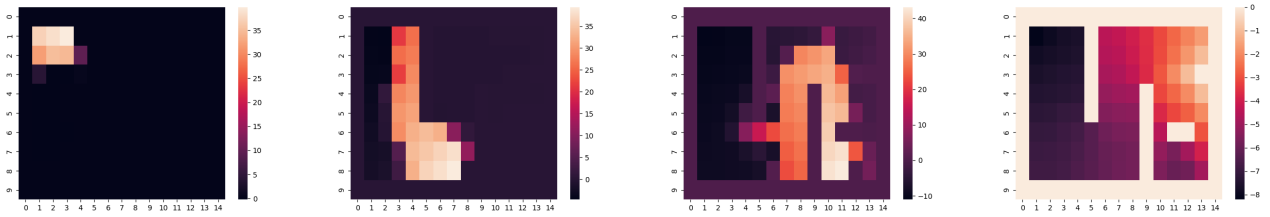
(a) C000 (b) C100 (c) C110 (d) C111

Figure 31: heat-maps experiment 4.4 for $\epsilon = 0.0$.



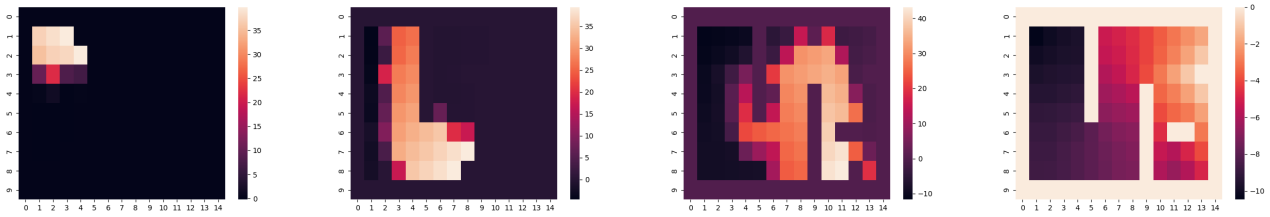
(a) C000 (b) C100 (c) C110 (d) C111

Figure 32: heat-maps experiment 4.4 for $\epsilon = 0.05$.



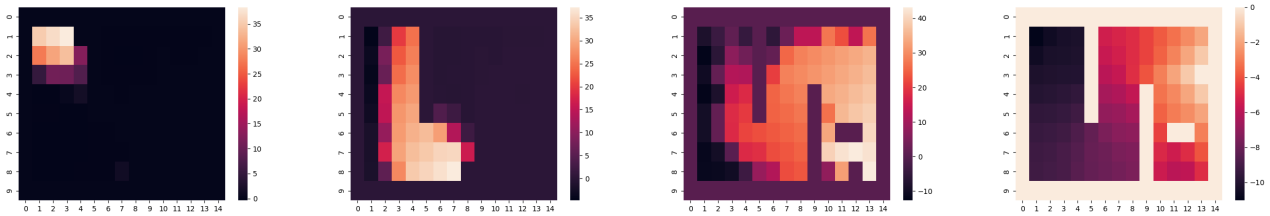
(a) C000 (b) C100 (c) C110 (d) C111

Figure 33: heat-maps experiment 4.4 for $\epsilon = 0.1$.



(a) C000 (b) C100 (c) C110 (d) C111

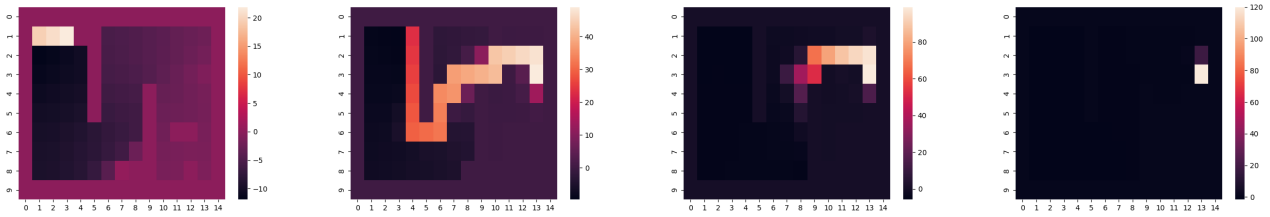
Figure 34: heat-maps experiment 4.4 for $\epsilon = 0.25$.



(a) C000 (b) C100 (c) C110 (d) C111

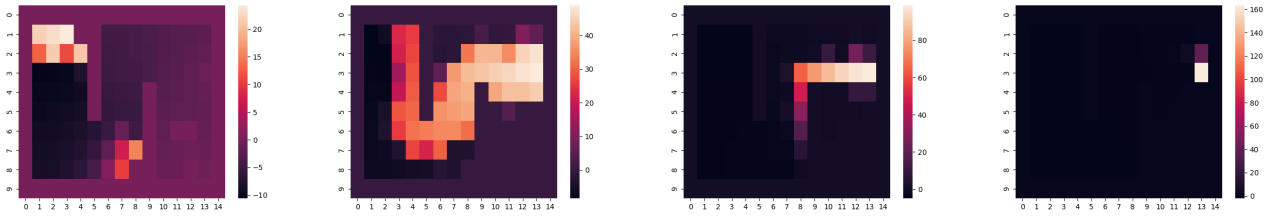
Figure 35: heat-maps experiment 4.4 for $\epsilon = 0.5$.

Experiment 4.5: Influence of late rewards
Cxxx indicates the (order of) picked up items for each situation.



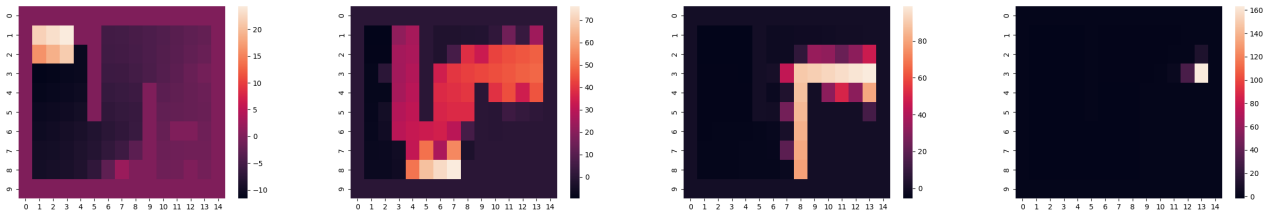
(a) C000 (b) C100 (c) C110 (d) C111

Figure 36: heat-maps experiment 4.5 for $\epsilon = 0.0$.



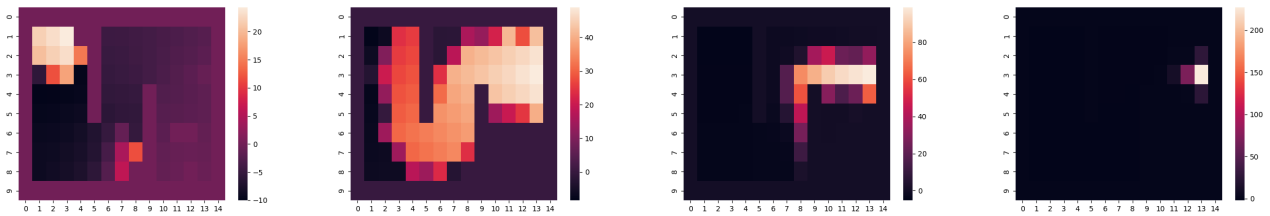
(a) C000 (b) C100 (c) C110 (d) C111

Figure 37: heat-maps experiment 4.5 for $\epsilon = 0.05$.



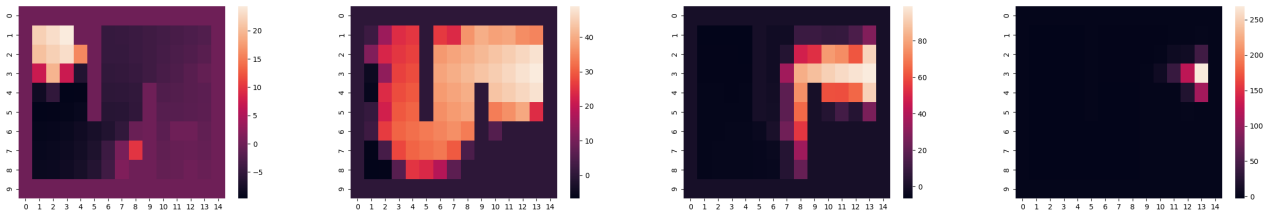
(a) C000 (b) C100 (c) C110 (d) C111

Figure 38: heat-maps experiment 4.5 for $\epsilon = 0.1$.



(a) C000 (b) C100 (c) C110 (d) C111

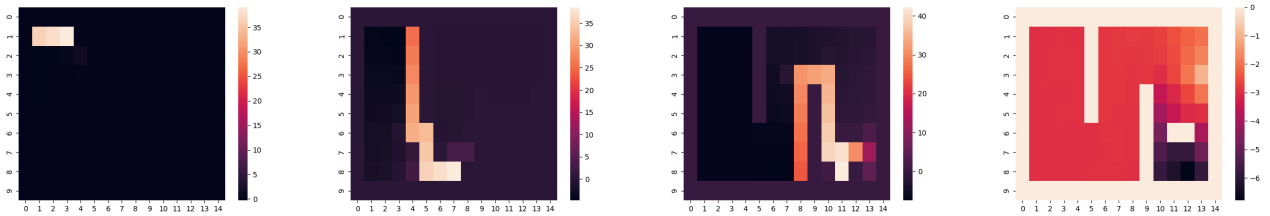
Figure 39: heat-maps experiment 4.5 for $\epsilon = 0.25$.



(a) C000 (b) C100 (c) C110 (d) C111

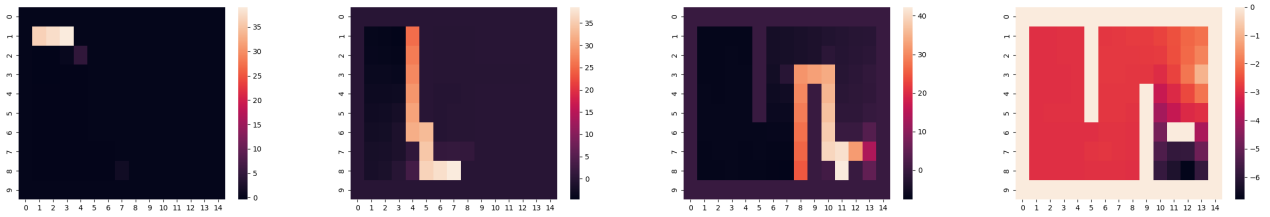
Figure 40: heat-maps experiment 4.5 for $\epsilon = 0.5$.

Experiment 4.6: Upper-Confidence Bound
Cxxx indicates the (order of) picked up items for each situation.



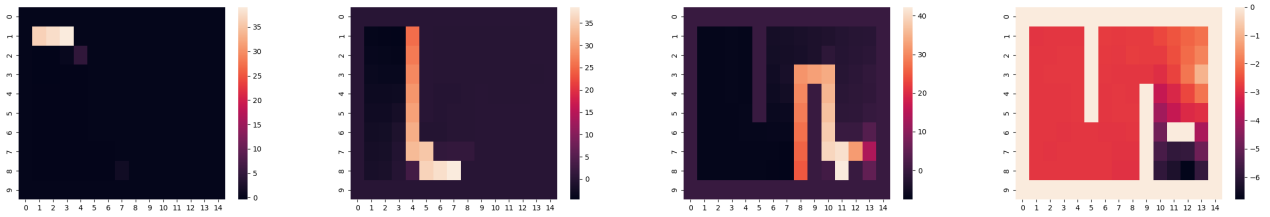
(a) C000 (b) C100 (c) C110 (d) C111

Figure 41: heat-maps experiment 4.6: full environment for $c = 0.0$.



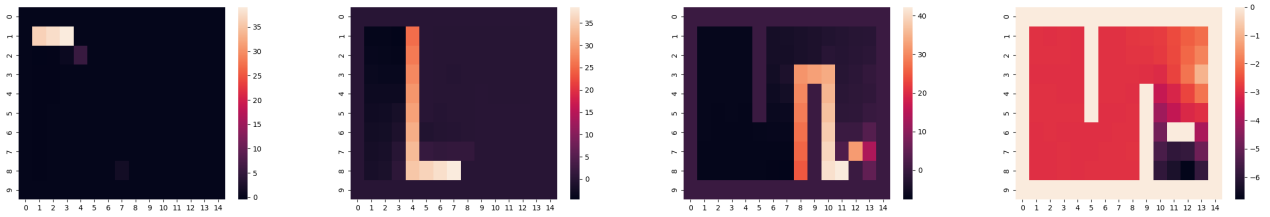
(a) C000 (b) C100 (c) C110 (d) C111

Figure 42: heat-maps experiment 4.6: full environment for $c = 0.05$.



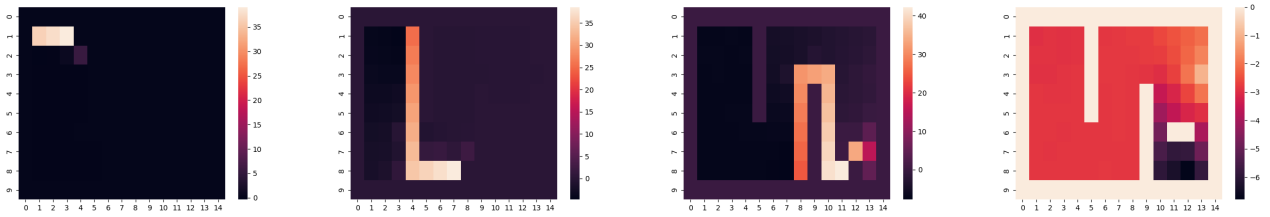
(a) C000 (b) C100 (c) C110 (d) C111

Figure 43: heat-maps experiment 4.6: full environment for $c = 0.1$.



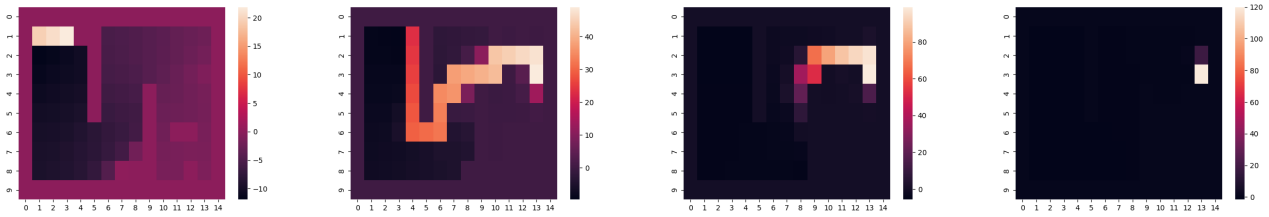
(a) C000 (b) C100 (c) C110 (d) C111

Figure 44: heat-maps experiment 4.6: full environment for $c = 0.25$.



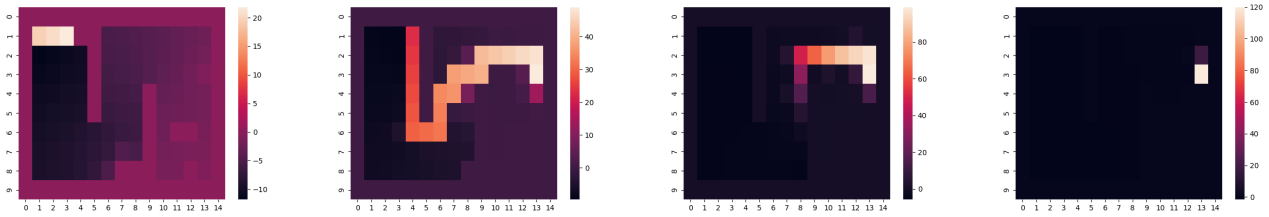
(a) C000 (b) C100 (c) C110 (d) C111

Figure 45: heat-maps experiment 4.6: full environment for $c = 0.5$.



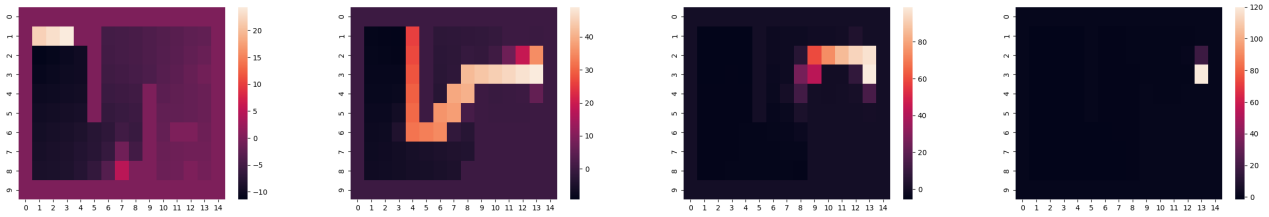
(a) C000 (b) C100 (c) C110 (d) C111

Figure 46: heat-maps experiment 4.6: late reward for $c = 0.0$.



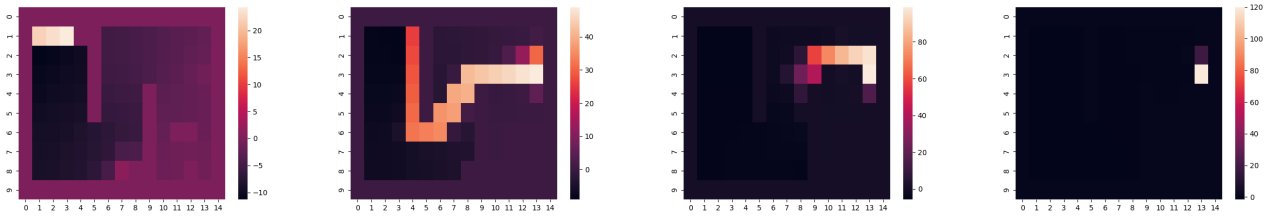
(a) C000 (b) C100 (c) C110 (d) C111

Figure 47: heat-maps experiment 4.6: late reward for $c = 0.05$.



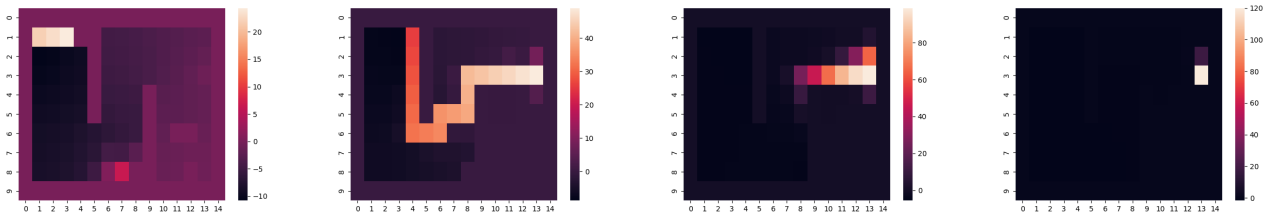
(a) C000 (b) C100 (c) C110 (d) C111

Figure 48: heat-maps experiment 4.6: late reward for $c = 0.1$.



(a) C000 (b) C100 (c) C110 (d) C111

Figure 49: heat-maps experiment 4.6: late reward for $c = 0.25$.



(a) C000 (b) C100 (c) C110 (d) C111

Figure 50: heat-maps experiment 4.6: late reward for $c = 0.5$.

B Experiment graphs

This appendix features three additional graphs obtained from the experiments.
Experiment 4.5: Influence of late rewards

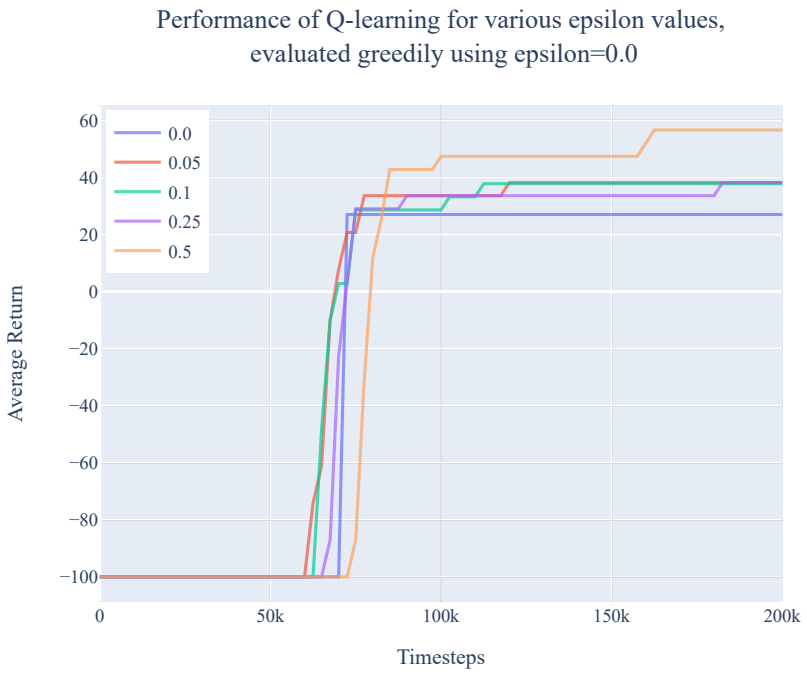


Figure 51: extra greedy evaluation plot with 200k time-steps, double the amount of in the experiment, showing that, mostly for $\epsilon = 0.5$, the algorithm is still improving after 100k time-steps under different ϵ values.

Experiment 4.6: Upper-Confidence Bound

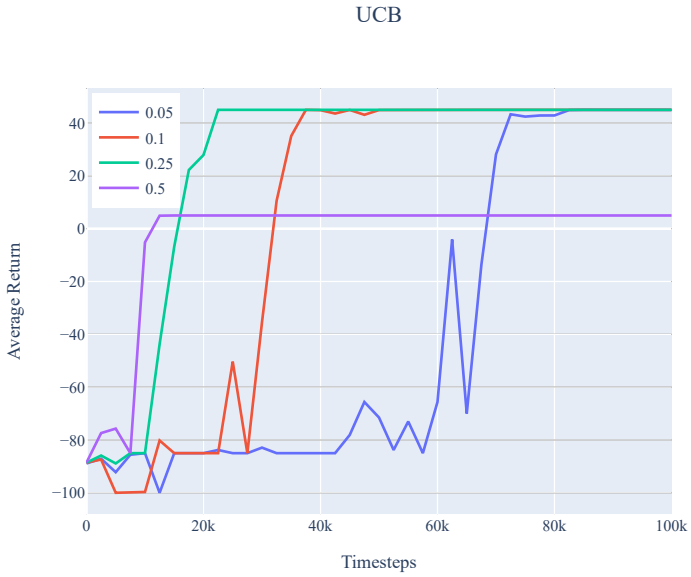


Figure 52: learning curves of Q-Learning using a UCB policy under different values for the learning rate α, using c=0.5 on the full environment

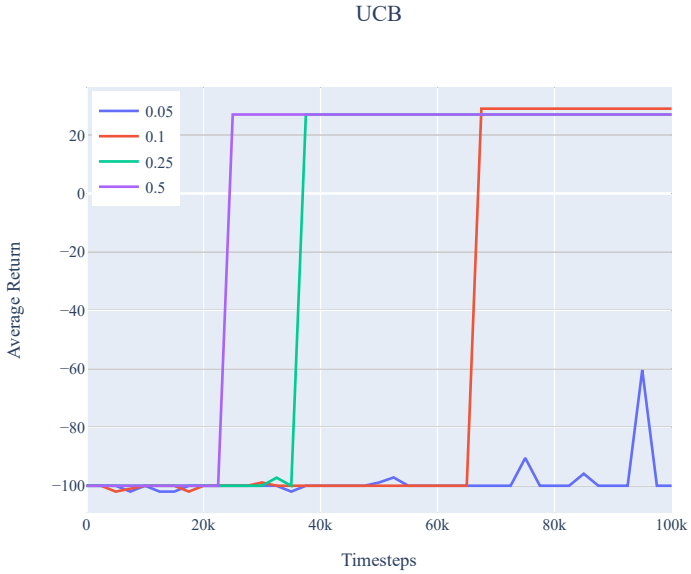


Figure 53: learning curves of Q-Learning using a UCB policy under different values for the learning rate α, using c=0.5 on the late reward environment