Reinforcement Learning Lab 1 Report

Eirini Stratigi 19940217-3844 stratigi@kth.se

Georgios Moschovis 19970325-7536 geomos@kth.se

Abstract

In this 1^{st} Lab we developed a maze with a Player and a Minotaur. We examined cases of how the Minotaur moves in order to decide the action of Player. The goal was to succeed exiting the maze alive. We developed a Markov Decision Process where we applied either Bellman equation or Value iteration, then we implemented Q-Learning and SARSA algorithms.

1 Problem: The Maze and the Random Minotaur

1.1 Basic maze

To formulate the problem as a Markov Decision Process (MDP) we used the following components.

State Space:

$\mathcal{S}_1 = \{(X_{pl}, Y_{pl}, X_{min}, Y_{min}), (X_{pl}, Y_{pl}) \neq$'Wall'$\}$, ($pl$: player, min : minotaur)

$\mathcal{S}_2 = \{$'Win','Lost' $\}$

$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$

Hereunder, we also describe the terminal states, which are either those the player is exiting the maze alive $\mathcal{S}_{Win}$ and those the player is eaten by the minotaur $\mathcal{S}_{Lost}$:

$\mathcal{S}_{Win} = \{(6, 5, X_{min}, Y_{min}), (X_{min}, Y_{min}) \neq (6, 5))\} \subset \mathcal{S}$

$\mathcal{S}_{Lost} = \{(X_{pl}, Y_{pl}, X_{min}, Y_{min}), (X_{pl}, Y_{pl}) = (X_{min}, Y_{min})\} \subset \mathcal{S}$

Actions:

$\mathcal{A} = \{$'Stay','Up','Down','Left','Right'$\}$

Rewards:
We used the following reward function taking into account the different cases illustrated below.

$$\mathcal{R} = r(s, a, s') = \begin{cases} 0, & \text{if } s' \in \mathcal{S}_2 = \{\text{'Win','Lost'}\} \\ 1, & \text{if } s' \in \text{S}_{Win} \\ -1, & \text{if } s' \in \text{S}_{Lost} \\ -10, & \text{if } s' \text{ : invalid state; outside grid or player hits wall} \\ -0.05, & \text{otherwise} \end{cases}$$

Transition Probabilities:
The Minotaur is choosing an action with probability $\frac{1}{N}$ and moves to a new position where N is the number of the possible moves that can be done at the cell that he is. For example, if he is located at the bottom right corner, thus he can move only up or left, he will do so with probability $\frac{1}{2}$. The player can move with probability 1 to his selected direction and with 0 to the other ones. For, example if his chosen action is 'left' he has zero probability to move to any other direction. Also, the minotaur and the player are moving in the grid independently and simultaneously. From these we conclude the following expression for the transition probabilities:

$$\mathcal{P}(s'|s,a) = \begin{cases} \frac{1}{N}, & \text{if } s' \text{ is accessible from } s, \\ 0, & \text{otherwise} \end{cases}$$

Training objective (for the basic maze):

$$\mathcal{O}(\mathcal{S},\mathcal{A},T) = \mathbb{E}\left[\sum_{t=0}^{T-1} r_t(s_t,a_t) + r_T(s_T)\right],$$

where $s_t \in \mathcal{S}\ \forall t \in [1, T-1] \cap \mathbb{N}$ and $a_t \in \mathcal{A}\ \forall t \in [1, T-1] \cap \mathbb{N}$.

1.2 Dynamic Programming

For the first part of this question we used the finite horizon MDP and run for $T = 20$. In this setting the player was always winning in $t = 15$ that means that he could **aways direct to the exit through the shortest path without being eaten by the minotaur before the allowed time**. A gif illustration of a simulation can be found in **here**. The exiting probability when the minotaur always moves is plotted below in Fig 1.

For the second part of this question we use the above finite horizon MDP and run for $T = 30$ but now allowing the minotaur to stay, which means that his allowed actions were increased by one, actually corresponding to him remaining in the same position. In this case we observe that the probability of exiting the maze increases over time, which we expected as if the player has the minotaur staying in front of him, he needs more time to escape from him. We provide another gif illustration of a simulation **here** and the exiting probability is plotted below in Fig 2.

Figure 1: Exiting probability when minotaur always moves

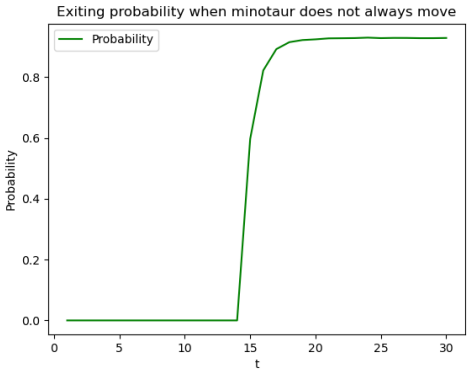

Figure 2: Exiting probability when minotaur does not always move.

1.3 Value Iteration

In this scenario we had that our agent's life was geometrically distributed with mean 30, therefore the decision maker has a time horizon T geometrically distributed and $\mathbb{E}[T] = \frac{1}{1-\lambda} = 30$. To solve an MDP with infinite horizon we use the value iteration algorithm after getting the value of λ:

$$\mathbb{E} = \frac{1}{1-\lambda} = 30 \Rightarrow \lambda = \frac{29}{30}$$

From the above we run value iteration for $\epsilon = 10^{-4}$ and $\lambda = \gamma = \frac{29}{30}$. We also provide another gif illustration of a simulation **here** as well. After simulating for 10^5 episodes we compute the average probability that the player escapes. We noticed this equals to 100% when the minotaur always moves and approximately to 88% when he can stay. The probability of exiting the maze over the 10^5 episodes is modelled as $\frac{\text{how many wins player had}}{\text{how many games they played}}$ and the drop when the minotaur can stay was expected because in cases where the player has the minotaur staying in front of him he might not be able to escape from him. For example consider the scenario where the minotaur is located at $(0, 5)$, the player at $(0, 6)$ and the minotaur always stays, then the player is blocked in a subarea of the maze where he can't escape from and will be eventually eaten by the minotaur.

1.4 Theoretical Questions

1.4.1 What does it mean that a learning method is on-policy or off-policy?

In off-policy learning whatever the behaviour policy is (even ϵ-greedy w.r.t. the currently estimated Q-function), $Q(t)$ converges to the true Q function. An example of an off-policy learning algorithm is the famous Q-learning algorithm, which in its updates always uses the max value of next state, in which case the (state, action) being taken to update the Q value may not be consistent with the current policy, thus it is called off-policy method. In this sense, Q-learning, that is off-policy learning, is being optimistic in value estimation, where it always assume the best action be taken in the process, which, could result in bolder actions of the agent. In off-policy learning the observations made by the agent along the trajectory of the dynamical system under the behavior policy π_b which should explore all actions.

On the other hand, on-policy learning is considered to be safer because it takes exploration into account. Precisely the update process of on-policy algorithms is consistent with the current policy, therefore behavior policy is safer, because on-policy learning optimizes the value of the behavior policy. An example of on-policy learning algorithm is SARSA. In the following subsections we may notice the difference between Q-learning and SARSA updates given in the respective procedures that we implemented for the maze. In on-policy learning the observations made by the agent in episode/step t are generated under the policy π_t that is supposed to converge to π^* and should explore all actions, at any time t.

1.4.2 State the convergence conditions for Q-learning and SARSA.

In theory it has been proved that Q-Learning will converge into an optimal solution but first we need to find which values of α and ϵ will give us this result; as we try for the maze problem below. Assume that the step sizes (α_t) satisfy $\sum_t a_t = \infty$ and $\sum_t a_t^2 < \infty$. Further assume that the behaviour policy π_b visits every (state, action) pairs infinitely often. Then for any discount factor $\in (0, 1)$,the Q-learning algorithm will converge almost surely; thus under the Q-learning algorithm, $\lim_{n\to\infty} Q^{(t)} = Q$ almost surely.

Some cases where the above conditions are satisfied include:

- if $a_t = \frac{1}{1+t}$ and the behaviour policy yields an irreducible Markov chain
- for the $\epsilon-$greedy behavior policy that selects an action uniformly at random with probability ϵ and $\arg\max_{a\in\mathcal{A}_{s_t}} Q^{(t)}(s_t, a)$ with probability $1 - \epsilon$

For SARSA, we hope that it will converge to an ϵ-soft policy $\bar{\pi}$ and action value function when it visits all the state-action pairs. The policy converges in the limit to greedy policy but this result has not being proved. Precisely assume that the step sizes (α_t) satisfy $\sum_t a_t = \infty$ and $\sum_t a_t^2 < \infty$.

Further assume that the policy π_t is the ϵ-greedy policy with respect to $Q^{(t)}$. For any discount factor $\in (0,1)$, the SARSA algorithm will converge to an ϵ-soft policy $\bar{\pi}$ almost surely; thus under the SARSA algorithm, $\lim_{n\to\infty} Q^{(t)} = Q^{(\bar{\pi})}$ almost surely.

1.5 Value iteration for the extended problem formulation

In the extended formulation we used again the value iteration algorithm, but now the agent's life was geometrically distributed with mean 50, therefore the decision maker has a time horizon T geometrically distributed and $\mathbb{E}[T] = \frac{1}{1-\lambda} = 50$. Solving this we get the value of λ below. Again we deal with an infinite horizon MDP but to deal with the additional requirements we need a new formulation for our state space $\mathcal{S}$, reward function R and transition probabilities, then by implementing accordingly the value iteration algorithm for the new MDP the resulting policy is illustrated in a gif of a simulation that can be found in **here**.

$$\mathbb{E} = \frac{1}{1-\lambda} = 50 \Rightarrow \lambda = \frac{49}{50}$$

We remark that for this scenario the MDP components are precisely modified as follows:

State Space:

$\mathcal{S}_1 = \{(X_{pl}, Y_{pl}, X_{min}, Y_{min}, \text{has keys}), (X_{pl}, Y_{pl}) \neq \text{'Wall'}\}$, has keys $\in$ { True, False} $= \mathcal{B}$

$\mathcal{S}_2 = \{\text{'Win','Lost' }\}$

$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$

We also describe the states where the player is exiting the maze alive $\mathcal{S}_{Win}$, he is eaten by the minotaur $\mathcal{S}_{Lost}$, and he gets the keys when he still does not have them $\mathcal{S}_{Keys}$:

$\mathcal{S}_{Win} = \{(6, 5, X_{min}, Y_{min}, \text{True}), (X_{min}, Y_{min}) \neq (6,5))\} \subset \mathcal{S}$

$\mathcal{S}_{Lost} = \{(X_{pl}, Y_{pl}, X_{min}, Y_{min}, \text{has keys}), (X_{pl}, Y_{pl})\text{=}(X_{min}, Y_{min})\} \subset \mathcal{S}$, has keys $\in \mathcal{B}$

$\mathcal{S}_{Keys} = \{(0, 7, X_{min}, Y_{min}, \text{False}), (X_{min}, Y_{min}) \neq (0,7))\} \subset \mathcal{S}$

Actions:

$\mathcal{A} = \{\text{'Stay','Up','Down','Left','Right'}\}$

Rewards:
We used the following reward function taking into account the different cases illustrated below. Notice that for the states belonging in $\mathcal{S}_{Keys} \subset \mathcal{S}$ we need to assign a high reward so that the agent is motivated to go there and get the keys. Also notice that in the way we now defined $\mathcal{S}_{Win}$ the agent needs to get the keys in order to get to a state belonging in this set $\mathcal{S}_{Win} \subset \mathcal{S}$ as the 'has keys' boolean flag needs to be true. **This flag is initialized as false and turns to true immediately when the agent visits a state belonging in $\mathcal{S}_{Keys}$ (and can never be turned to false again)**. This will also be well noticed in the transition probabilities.

$$\mathcal{R} = r(s,a,s') = \begin{cases} 0, & \text{if } s' \in \mathcal{S}_2 = \{\text{'Win','Lost'}\} \\ 1, & \text{if } s' \in \mathcal{S}_{Win} \\ -1, & \text{if } s' \in \mathcal{S}_{Lost} \\ 10, & \text{if } s' \in \mathcal{S}_{Keys} \\ -10, & \text{if } s' \text{ : invalid state; outside grid or player hits wall} \\ -0.05, & \text{otherwise} \end{cases}$$

Transition Probabilities:
In this scenario we get that the minotaur is moving directly towards the player with probability 35% and uniformly at random with the remaining probability 65%. In this context, as well as to incorporate the extension of the states with the boolean flag 'has keys' indicating whether the player has the keys or not, we had to change the transition probabilities as follows. As for measuring which state s' that is accessible from s given the player's action a models the closest approach of the minotaur towards the player we use the Manhattan Distance (or L_1 distance) between them that is defined as $L_1(pl, min) = |X_{pl} - X_{min}| + |Y_{pl} - Y_{min}|$.

$$\mathcal{P}(s'|s,a) = \begin{cases} 0.65 \times \frac{1}{N-1}, & \text{if } s' \text{ is accessible from } s \text{ and is does not model the closest approach of the minotaur towards the player} \\ 0.35, & \text{if } s' \text{ is accessible from } s \text{ and models the closest approach of the minotaur towards the player} \\ 1, & \text{if } s' = (0,7,x,y,\text{True}), \text{ where } (x,y) \neq (0,7) \text{ and } s = (0,7,x,y,\text{False}) \\ 0, & \text{otherwise} \end{cases}$$

We first test that our code actually implements the transition probabilities demonstrated above by setting the probability of the minotaur approaching (stating) the player to 95% (instead of 35%) and of course each of the remaining states has probability $0.05 \times \frac{1}{N-1}$. We see that in this scenario the minotaur becomes very hungry and chases the player as illustrated in this gif in here. Finally we incorporate the keys concept and the final solution (simulation) where **the agent first gets the keys and then exits the maze alive** is illustrated in another gif that can be found in here. This highlights the correctness of our MDP formulation and our implementation.

1.6 Q-Learning algorithm (bonus)

In this section we describe how we solved the problem in its extended formulation introduced in section 1.5 using the famous Q-learning algorithm. **We include a pseudocode of our implementation of the Q-learning algorithm in the last page of the report.** We try out two different initialization strategies, one setting $Q(s,a) = 0 \; \forall s \in \mathcal{S}, a \in \mathcal{A}$ and one drawing the Q-values as random from a uniform distribution $Q(s,a) \in (0,1) \sim \mathcal{U} \; \forall s \in \mathcal{S}, a \in \mathcal{A}$. We denote the two strategies as 'Zeros', 'Random Uniform' and the value functions and probability of exiting the maze when simulating for 10^5 episodes are both illustrated below.

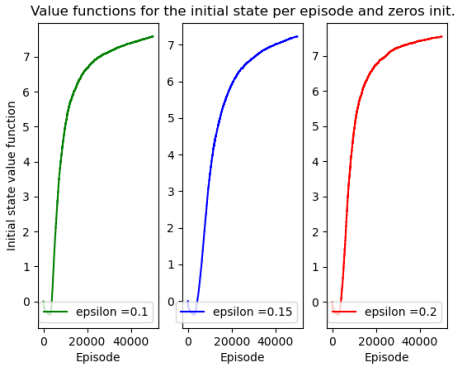

Figure 3: Value functions for the initial state per episodes and zeros init., $\epsilon \in \{0.1, 0.15, 0.2\}$

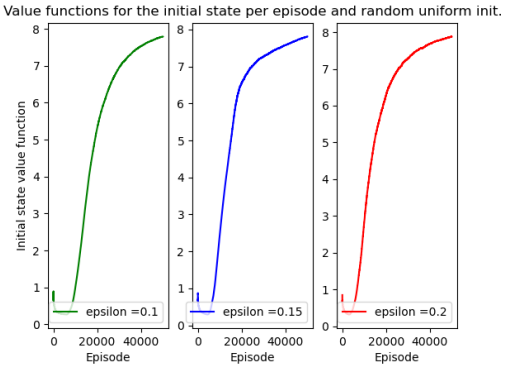

Figure 4: Value functions for the initial state per episode and random init., $\epsilon \in \{0.1, 0.15, 0.2\}$

Here we notice how a proper initialization of the Q-values actually affect convergence speed but also matters in the convergence of the algorithm (zeros init. illustrated in green gives a lot better performance than random uniform init. illustrated in blue in Fig. 10 in the next page). We realize the convergence speed by examining when the value function $v(s_0) = \arg\max_{a \in \mathcal{A}} Q(s_0, a)$ of the initial state $s_0 = (0, 0, 5, 6, \text{False})$ reaches a plateau. Moreover, we realize the performance of the algorithm as the probability of exiting the maze over the 10^5 episodes that is $\frac{\text{how many wins player had}}{\text{how many games they played}}$ and gets its maximum value for $\epsilon = 0.1$ and zeros initialization (the green point at the top right of Fig. 10 at the next page).

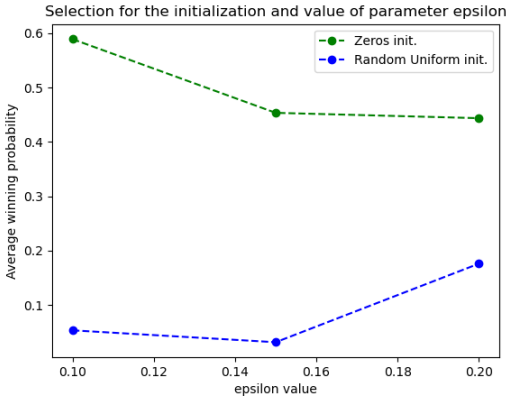

Figure 5: Selection for the initialization and value of parameter $\epsilon \in \{0.1, 0.15, 0.2\}$

By comparing the algorithm performance difference for the two initialization strategies we catered for, hereunder we will only focus on initializing the Q-values all to zero $Q(s, a) = 0 \, \forall s \in \mathcal{S}, a \in \mathcal{A}$, as this recipe achieves significantly higher performance. Then we experiment with different values of the parameter α that affects the learning rate (step size) $lr = \frac{1}{n(s,a)^{\alpha}}$ for a fixed value of $\epsilon = 0.1$ that gave the best performance as illustrated in Fig. 10 above. We notice that a large value of $\alpha = 1$ makes the learning impossible as the algorithm fails to converge, as illustrated in the red curve of figure 6 below, while $\alpha = 0.55$ (green curve) reaches a plateau faster than $\alpha = 0.75$ (blue curve) in the same Fig. 6. However, the best performance is obtained by the default value of $\alpha = \frac{2}{3}$ as illustrated by second green point in Fig. 7 and the respective value function plot corresponds to the green curve in Fig. 8 in the previous page.

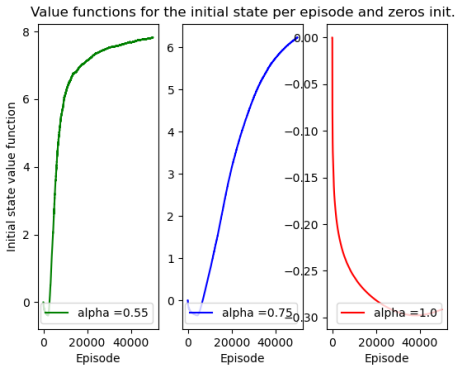

Figure 6: Value functions for the initial state per episodes and zeros init., $\alpha \in \{0.55, 0.75, 1\}$

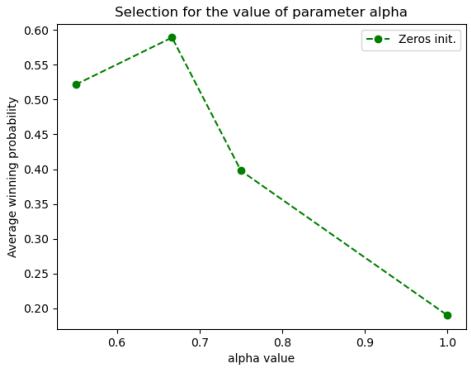

Figure 7: Selection for the value of parameter $\alpha \in \{0.55, \frac{2}{3}, 0.75, 1\}$

SARSA algorithm (bonus)

In this section we describe how we solved the problem in its extended formulation introduced in section 1.5 using the SARSA (Step-Action-Reward-Step-Action) algorithm. **We include a pseudocode of our implementation of the SARSA algorithm in the last page of the report.** We try out two different initialization strategies, one setting $Q(s, a) = 0 \, \forall s \in \mathcal{S}, a \in \mathcal{A}$ and one drawing the Q-values as random from a uniform distribution $Q(s, a) \in (0, 1) \sim \mathcal{U} \, \forall s \in \mathcal{S}, a \in \mathcal{A}$ exactly as we did with the Q-learning algorithm in the previous section. We denote the two strategies as 'Zeros', 'Random Uniform' and the value functions and probability of exiting the maze when simulating for 10^5 episodes are both illustrated in the following page. However, we notice that SARSA yields a very poor performance and fails to converge for the respective number of episodes, compared to Q-learning that achieved significant performance. However, Q-learning was an off-policy algorithm, while SARSA is an on-policy algorithm.

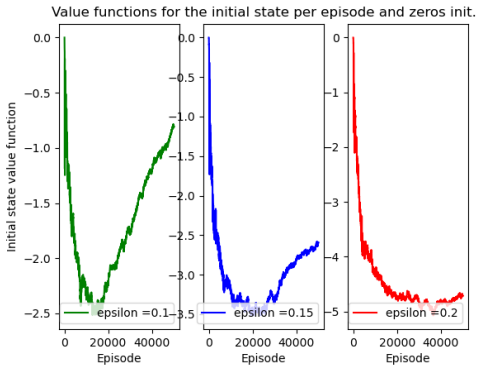

Figure 8: Value functions for the initial state per episodes and zeros init., $\epsilon \in \{0.1, 0.15, 0.2\}$

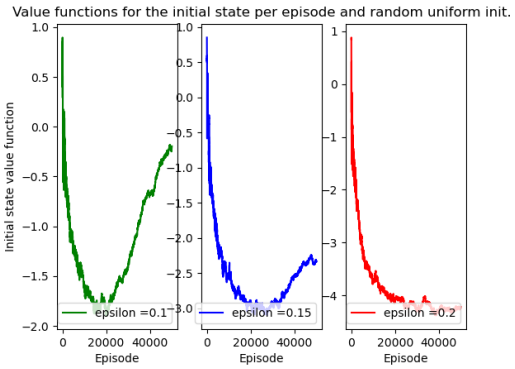

Figure 9: Value functions for the initial state per episode and random init., $\epsilon \in \{0.1, 0.15, 0.2\}$

We tried to identify the reasons why SARSA fails to achieve as good performance as Q-learning and we mainly believe that is is related to the fact that the former performs on-policy learning, therefore the update process is consistent with the current policy, while the latter performs off-policy learning and always uses the max value of next state, in which case the state, action being taken to update the Q value may not be consistent with the current policy, thus it is called off-policy method. In this sense, Q-learning (off-policy) is being optimistic in value estimation, where it always assume the best action be taken in the process, which, could result in bolder actions of the agent that is a desired effect in our setting.

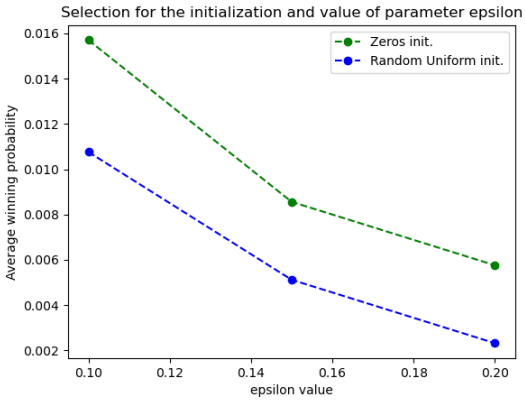

Figure 10: Selection for the initialization and value of parameter $\epsilon \in \{0.1, 0.15, 0.2\}$

Then we tried to deal with SARSA failing to achieve as good performance as Q-learning by replacing the constant values for ϵ with a formula computing epsilon as a function of the episode k, i.e. $(k, \delta) = \frac{1}{k^\delta}$, where δ is a predefined constant. Once again we experimented with different combinations of values for the parameters α, δ, some among which improved the performance of SARSA algorithm. That is, for each of the values of $\alpha \in \{0.55, \frac{2}{3}, 0.75, 1\}$ we experimented also with $\delta \in \{0.55, 0.75, 1\}$, among which combinations the best performance was obtained for $\alpha = \frac{2}{3}$, $\delta = 1$, that is illustrated in figures 12, 14, 16, 18 (the green point at the top right of Fig. 14 corresponds to the best model).

Again performance is measured as the probability of exiting the maze over the 10^5 episodes that is $\frac{\text{how many wins player had}}{\text{how many games they played}}$. The learning rate is also still defined as $lr = \frac{1}{n(s,a)^\alpha}$ but the value of ϵ is determined by $\epsilon_k = (k, \delta) = \frac{1}{k^\delta}$ and in the best scenario $\alpha = \frac{2}{3}$, $\delta = 1$, thus $\alpha < \delta$. This inequality holds also for most scenarios illustrated in figures 12, 14, 16, 18, that is performance is generally better when $\alpha < \delta$. Apart from the probability of exiting the maze we report again value function curves for the initial state $s_0 = (0, 0, 6, 5, \text{False})$ in figures 11, 13, 15, 17.

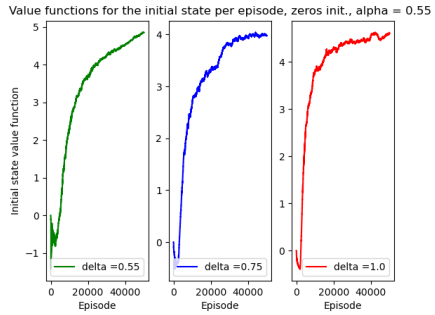

Figure 11: Value functions for the initial state per episodes, zeros init., $\alpha = 0.55$, and $\delta \in \{0.55. 0.75. 1\}$

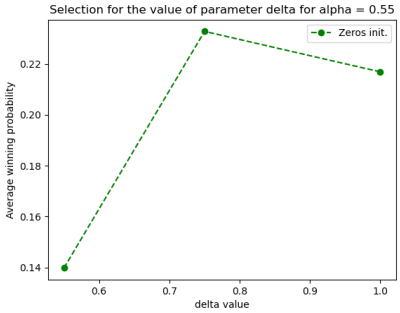

Figure 12: Selection for the value of the parameter $\delta \in \{0.55, 0.75, 1\}$ for a constant value of $\alpha = 0.55$

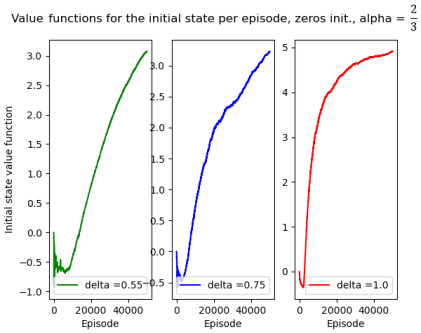

Figure 13: Value functions for the initial state per episodes, zeros init., $\alpha = \frac{2}{3}$, and $\delta \in \{0.55. 0.75. 1\}$

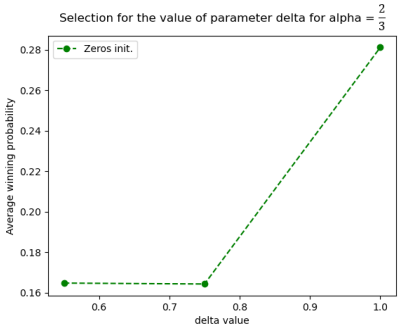

Figure 14: Selection for the value of the parameter $\delta \in \{0.55, 0.75, 1\}$ for a constant value of $\alpha = \frac{2}{2}$

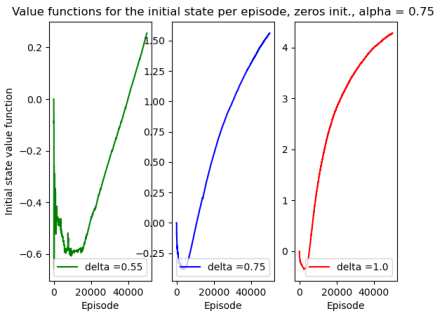

Figure 15: Value functions for the initial state per episodes, zeros init., $\alpha = 0.75$, and $\delta \in \{0.55. 0.75. 1\}$

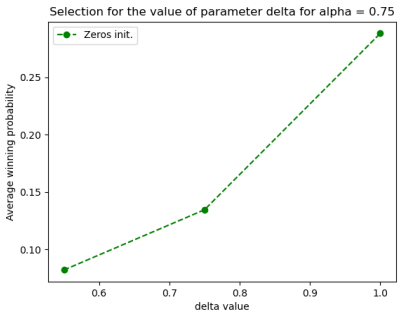

Figure 16: Selection for the value of the parameter $\delta \in \{0.55, 0.75, 1\}$ for a constant value of $\alpha = 0.75$

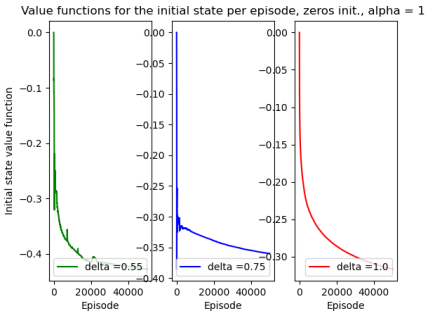

Figure 17: Value functions for the initial state per episodes, zeros init., $\alpha = 1.0$, and $\delta \in \{0.55, 0.75, 1\}$

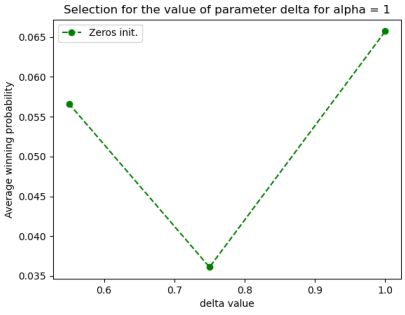

Figure 18: Selection for the value of the parameter $\delta \in \{0.55, 0.75, 1\}$ for a constant value of $\alpha = 1$