Solving mazes with Q-learning

Konstantinos Theofilos Drylerakis
Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
ktd1g20@soton.ac.uk

***Abstract*—A dynamic maze problem is solved with the aid of a Q-learning algorithm. To aid in the process and accelerate the training phase, a dynamic reward function is utilised along with a framework that exploits past information collected in previous attempts of solving the maze. The challenges arising under this strategy are discussed, alternative solutions are proposed and the limitations of the proposed approach towards generalising the results are highlighted.**

I. Introduction

The following document constitutes a report about the approaches that were examined towards solving a maze problem using reinforcement learning. More specifically, Q-learning was used towards solving the maze. The structure of the report is as follows:

- Section II provides a background discussion about the algorithms used, the problem description and the software used
- Section III provides the methodology of the approach adopted
- Section IV provides the results of the proposed approach
- Section V provides a discussion about the results, possible alternative solutions considered, as well as the limitations of the proposed approach
- Section VI concludes the report
- Section VII describes how to compile and run the code that accompanies this report. The relevant files can also be found through the following github link: https://github.com/DKT91?tab=repositories

II. Background

In this section the reinforcement learning algorithms used towards solving the maze problem are presented. The algorithms are drawn from [1] and [2]. Furthermore a short description of the maze and the problem at hand are given. Finally, the software used is also mentioned for reproducibility purposes.

A. Double Q-learning

Algorithm 1 corresponds to the tabular case of Q-learning examined in this work. Q^A and Q^B correspond to the two Q-tables used, s and s' represent the previous and next states respectively, a belongs to the set of actions, α corresponds to the learning rate and γ is the discount factor. The immediate reward is represented by r. In the case that only one Q-table is used, the two Q-tables of this algorithm are identical and refer to the same object from a programming perspective, and therefore the algorithm stands as is for the simple case of tabular Q-learning.

Algorithm 1 Double Q-learning

Initialise Q^A, Q^B, s
repeat
  Choose a based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = argmax_a Q^A(s',a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)(r + \gamma Q^B(s',a^*) - Q^A(s,a))$
  else if UPDATE(B) **then**
    Define $b^* = argmax_a Q^B(s',a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
  end if
  $s \leftarrow s'$
until end

B. Deep reinforcement learning with Double Q-learning

Algorithm 2 corresponds to the deep learning case of Q-learning examined in this work. The weights of the online and target network are represented by $\mathbf{w}$ and $\mathbf{w}'$ respectively. The current and next state are represented as s and s' respectively. The immediate reward is represented as r and the reward for an element in the experience buffer is represented as $\hat{r}$. An action for an element in the experience buffer is represented as $\hat{a}$ and the selected action for the current state s as a. The discount factor is represented as γ and the learning rate as α.

C. Maze and problem description

The maze can be viewed as a 201x201 array and can be further depicted as the illustration in Figure 1. The outline of the maze corresponds to walls only. An agent is placed on the top left tile of the maze i.e., location $(1,1)$ in (x,y) coordinates representing the latitude and longitude of the position respectively. The objective is for the agent to travel to the bottom right tile i.e., location $(199,199)$ of the maze, in the minimum time possible. In the case examined in this work, minimum traversal time and minimum traversal path towards the $(199,199)$ tile are equivalent, as it is assumed that each move of the agent takes the same amount of time.

Algorithm 2 Deep reinforcement learning with Double Q-learning

Initialise weights $\mathbf{w}$, $\mathbf{w}'$ at random in $[-1, 1]$
Observe current state s
repeat
  Select action a and execute it
  Receive immediate reward r
  Observe new state s'
  Add (s, a, s', r) to experience buffer
  Sample mini-batch of experiences from buffer
  for each experience $(\hat{s}, \hat{a}, \hat{s}', \hat{r})$ in mini-batch **do**
    Gradient:

$$\frac{\partial Err}{\partial \mathbf{w}} = [Q_{\mathbf{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma max_{\hat{a}'} Q_{\mathbf{w}'}(\hat{s}', \hat{a}')] \frac{\partial Q_{\mathbf{w}}(\hat{s}, \hat{a})}{\partial \mathbf{w}}$$

    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial Err}{\partial \mathbf{w}}$
  end for
  Update state: $s \leftarrow s'$
  Every c steps, update target $\mathbf{w}' \leftarrow \mathbf{w}$
until end

The agent may only move left, up, right, down or remain at the same position. The agent has to observe its local surroundings before taking an action at each step of the process. This is done by getting information regarding the 8 neighbouring tiles in terms of the presence of fire and walls and is received by the agent in the form of two 3x3 matrices, one for the information regarding walls and one for the information regarding fire. The agent is assumed to be in the center of these two matrices when they are regarded as grids. However, getting local information may result in fires induced at some neighbouring locations of the agent's current position. The mechanism through which fires appear is random and therefore fires cannot be predicted. The agent cannot move into fires and has to act in some other way in order to either avoid the fire and continue, or wait until it has been extinguished by observing its surroundings again in the next step. Therefore, this is a case of a dynamic maze solving problem.

D. Software used

All software was run on Google Colab using a Jupyter Notebook environment with default settings.

III. Methodology

In order to solve the problem at hand the algorithms presented in Section II were used. However, further work was done towards accelerating the process. The notions of "bad states", "bad intersections" and "bad intersection moves" are presented in this section as well as the overall structure of the proposed solution along with the definition of the reward function and the space of states.

A. Bad states

The name "bad states" simply refers to tiles of the maze of Figure 1 that correspond to dead ends. Such a state can be defined as the state that when observed by the agent, only one possible move exists in order to progress to another state when fire is not taken into account. In other words, such a state corresponds to reaching a dead end. An example is given in Figure 2. If the agent ends up in a position similar to $(1, 1)$ in Figure 2 then the state it has reached is bad.

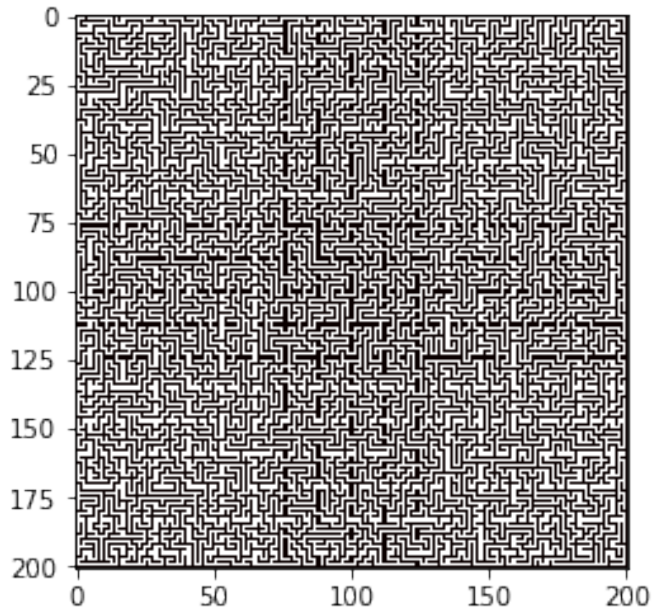

Fig. 1. The maze to solve represented as a binary heat map. Black sections are walls while white sections are potentially empty spaces, conditioned on the distribution of fires.

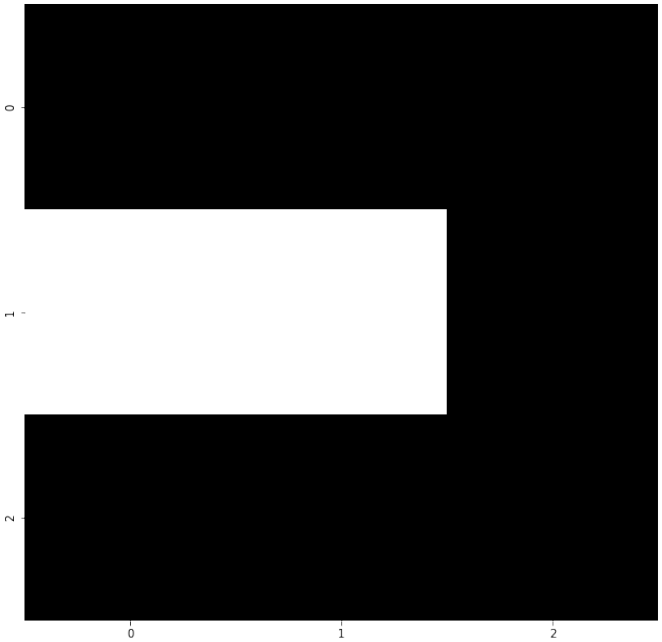

Fig. 2. An example of bad state. If the agent reaches position $(1, 1)$ then only one move is possible to alter its position if fire is disregarded.

B. Bad intersection moves

The name "bad intersection moves" simply refers to moves made at intersection points of the maze that would lead towards bad states. Intersections are considered as locations in the maze that at least three moves are possible when fire is not taken into account. An example is given in Figure 3. If the agent was initially positioned on the $(1, 1)$ tile and made a move to the right this would result in a new state that is closer to the bad state located at the $(1, 3)$ location of the same figure. Therefore, making a right on tile $(1, 1)$ of Figure 3 is a bad intersection move and should be considered redundant, unless of course the agent's objective is to reach tile $(1, 3)$.

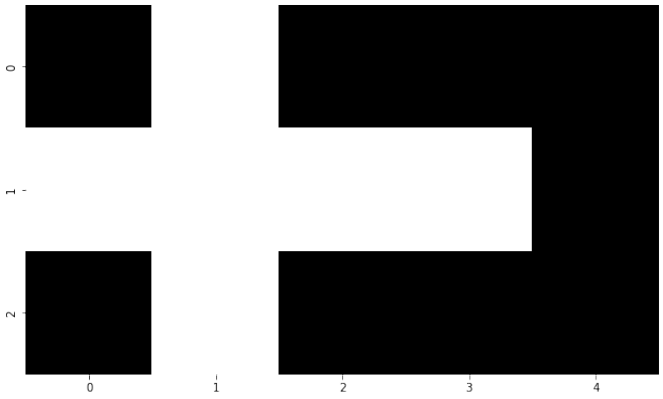

Fig. 3. An example of a bad intersection move would be to make a right when reaching the intersection tile $(1, 1)$. This is a bad intersection move as it leads to a bad state and the agent should be discouraged from making such moves.

C. Bad intersections

The name "bad intersections" simply refers to intersection points of the maze where only one move made from that point does not lead to a bad state. An example is given on Figure 4. The only move of the agent that would not lead to a bad state is making a left when reaching tile $(, 1)$. Therefore, reaching such an intersection is redundant and more specifically, bad intersections are equivalent to bad states, as in essence a bad intersection is a dead end.

D. Structure of solution

In order to speed up the training process for the agent, the notions of "bad states", "bad intersection moves" and "bad intersections" were used. More specifically, when the agent reaches a dead end, it records the intersection and corresponding move that led to it. This would be the last intersection it crossed and the move it made on it. Once that tuple of (intersection latitude, intersection longitude, intersection move) is added to the set of "bad intersection moves", it is checked whether that last intersection the agent crossed has turned bad. This is done via checking how many moves at that intersection are in the set of "bad intersection moves". If three moves are in this set then the intersection has turned bad and is added in the set of "bad states" (as bad intersections

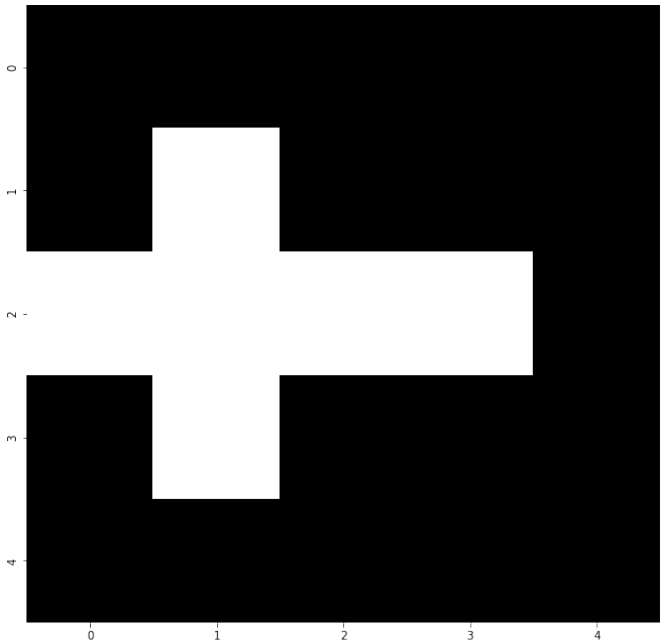

Fig. 4. An example of a bad intersection is tile $(2, 1)$. This is a bad intersection as there is only one move that can lead to a state that is not bad, which in that case would be going left.

are equivalent to bad states) and the agent is reset on some other intersection that has not turned bad yet. If however, that intersection does not end up in the set of "bad states" the agent is reset on that last intersection and continues from that point. It should be noted that walking into a wall at intersections is also considered a bad move and it belongs to the set of "bad intersection moves" by definition. Also, it must be noted that in this work the starting $(1, 1)$ point of the maze is considered an intersection as well and it is the only intersection that is not considered bad while allowing only one good move to be made. This modification allows the proposed approach to work right from the starting $(1, 1)$ tile of the maze. The overall process is illustrated in Figure 5.

E. Reward function and space of states

Many experiments were conducted regarding the rewards for the agent. Overall, however, the rewards could be structured in any way in terms of the actual numbers used, as long as the following strategy remains the same in principle:

- The agent gets a large negative reward when walking into a wall or a fire
- The agent gets a large negative reward when making a move that belongs to the set of "bad intersection moves"
- The agent gets a large negative reward when reaching a state that is in the set of "bad states"
- The agent gets a small negative reward when reaching new tiles in order to encourage the agent to explore the maze
- The agent gets a high positive reward when reaching the final $(199, 199)$ tile

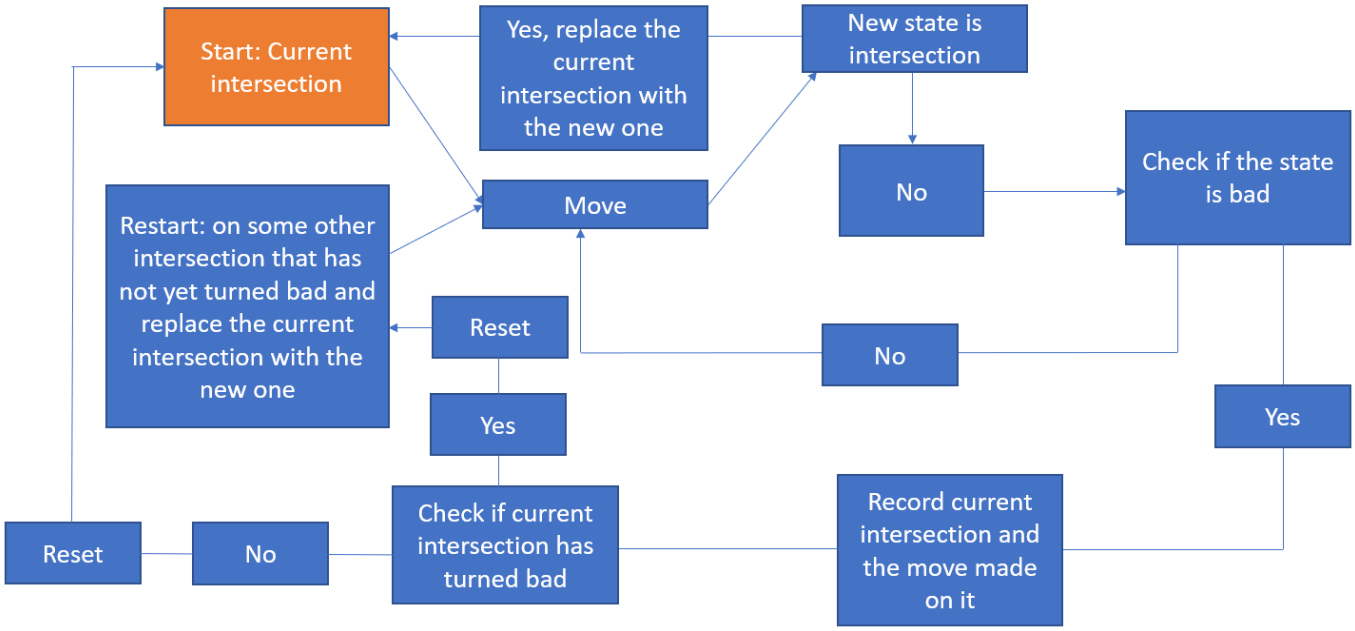

Fig. 5. The overall sutructure of the proposed solution towards solving the maze and accelerating the training process.

In terms of the space of states, four cases were considered to facilitate solving the maze:

- In the case of a static maze with no fire using tabular Q-learning, the state space was a set of latitude and longitude tuples e.g., $(3, 5)$. The agent only needs these two coordinates to decide on which move to make. Therefore the Q-table could be represented as a $(201, 201, 4)$ array in Python notation, with each element corresponding to the Q-value Q(s,a) where s is a (x,y) tuple representing the state and a is a number representing which of the four possible moves is being made. Therefore, for the case of a static maze only moves left, up, right and down were considered as there is no point for the agent to remain at the same location when fire is not taken into account.
- In the case of a static maze with no fire using a deep Q-network, the state space was a set of latitude and longitude tuples similar to the case of the tabular Q-learning algorithm. Therefore, the inputs to the network are the coordinates of the location of the agent. The output is a tuple containing the Q-values for each of the four possible actions at this state so that the one with the highest Q-value can be executed. Like in the tabular case of the static maze, only four actions are considered as there is no point for the agent to remain at the same location.
- In the case of a dynamic maze with fire using tabular Q-learning, the state space was a set of tuples containing the latitude, longitude and information about the fire. In order to take into account the fire, four binary variables were introduced: fire left, fire up, fire right and fire down. A value of 1 for these variables corresponds to fire being present towards that direction in relevance to the agent's current location. A value of 0 corresponds to the absence of fire towards that direction in relevance to the agent's current location. It should be noted that while this is an augemented state space when compared to the one used in the static case of the maze, it fails to fully represent all the possible states as in the dynamic case of the maze a state with e.g., a fire that will remain for 2 time steps in the direction above the agent is different to the state with the same coordinates but with a fire that will remain for 3 time steps in the direction above the agent's current position. Therefore the state space introduced here only captures information pertaining to the presence of fire and not the duration of it. This space can then be represented as a $(201, 201, 2, 2, 2, 2)$ array in Python notation and the corresponding Q-table would be a $(201, 201, 2, 2, 2, 2, 5)$ array with each element representing the Q-value Q(s,a) where s is a (x,y,f_1,f_2,f_3,f_4) tuple representing the state and a being a number representing which of the five possible moves is being made (left, up, right, down, remain).
- In the case of a dynamic maze with fire using a deep Q-network, the state space was a set of latitude and longitude tuples along with the local information around the agent. Since, at each step the agent observes its local environment, the inputs to the network are the coordinates of the location of the agent along with the information of its local environment, therefore corresponding to a 20-dimensional vector (the elements of the two 3x3 matrices capturing the local information and the latitude and longitude coordinates of the agent's position). The output is a tuple containing the Q-values for each of the five possible actions at this state.

IV. Results

A. Static maze using tabular Q-learning

The approach of using two Q-tables, in accordance with Algorithm 1, was examined. However, it was observed that while using two Q-tables can lead to better estimations of the Q-values, the overall training process is slowed down significantly considering that the problem at hand is not especially complex. Therefore, as the challenges that emerged from using a double Q-learning approach out weighted the overestimation issues of using a single Q-table, a single-table Q-learning approach was used eventually.

In the case that fire is disregarded when traversing the maze, it takes about 3 hours to completely scan the maze using the framework presented in Figure 5. After this initial run the algorithm is able to reach the (199,199) tile in about 4 minutes in training mode. The discount factor γ was set to 0.999 while the learning rate α was set to a constant rate of 0.1 for the initial parts of the training phase. After a certain point in training mode the set of "bad intersection moves" is finalised. Once this is achieved, this set can be used to retrain the model from the start. This was done to accelerate the convergence of the Q-table. In order to even further accelerate the convergence of the algorithm, the learning rate was set to $n(s,a)^{-1000}$ where $n(s,a)$ is the number of times that action a was taken at state s. Therefore, the Q-value $Q(s,a)$ is essentially updated only the first time that action a is taken on state s. Combined with the information received from the previously found set of "bad intersection moves" and the way that the reward function was formulated, this setting resulted in a fast convergence rate for the model. Training the algorithm for 4 hours was enough for the Q-table to converge and from that point on the maze can be solved in testing mode, in which the Q-table's values are not further updated. The extracted path can be found in Figure 6 and it corresponds to 3,584 moves made by the agent.

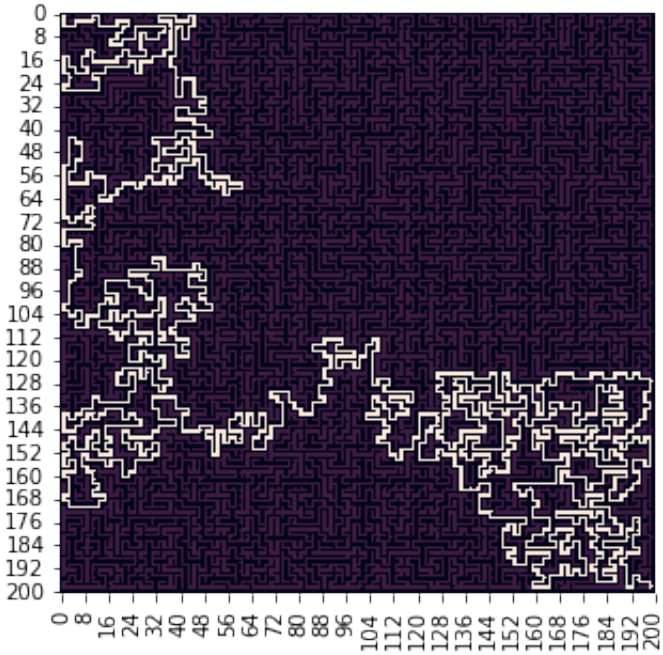

Fig. 6. The extracted trajectory of the agent in the case of a static maze with no fire using tabular Q-learning.

B. Dynamic maze using tabular Q-learning

As in the case of the static maze using tabular Q-learning, there were issues with using two Q-tables. Therefore, a single Q-table was used eventually as the objective was to compute the best path towards the (199,199) tile and not to optimise the estimation for the Q-values.

A similar approach to the one used for the static case of the maze in section IV.A was used for the corresponding dynamic one as well. The discount factor γ was set to 0.999 while the learning rate α was set to a constant rate of 0.1 for the initial parts of the training phase. In this case however, the algorithm takes significantly more time to completely scan the maze. This is due to the dynamic nature of the problem resulting in a much larger size for the space of states. Furthermore, there are occasions where the agent can not move due to fire emerging in neighbouring cells. Nevertheless, after the 10 hour mark the agent will consistently reach the (199,199) tile in training mode in about 8 minutes, depending on the stochastic elements of each individual run. Unlike the static case of the maze however, it is impossible to tell before hand how long the model needs to train to ensure that the agent can solve the maze for every scenario. This is because it can not be ensured that the agent will go through all possible states of the Q-table as the states that the agent will find itself in are random. To address this problem, the model was, as in the case examined in section IV.A, retrained using the extracted set of "list of bad intersection moves" and a learning rate of $n(s,a)^{-1000}$ where $n(s,a)$ is the number of times that action a was taken at state s. Training the model in such a way for 200 hours however, was not enough to produce a Q-table that is able to solve the maze in evaluation mode for the general case scenario. More specifically, the extracted Q-table was tested on 100 consecutive trials of traversing the maze and the success rate was zero. Therefore, it should be noted that on most occasions the agent is not able to go through the maze in evaluation mode currently. To solve that issue, more training is required for the model. A successful run for the dynamic case is depicted on Figure 7, which corresponds to the contents of the accompanying output file of this work. The number of moves needed for the agent were 4,133. However, this working solution is based on using the Q-values for the static case of the maze and will be explained further in section V.

C. Static and dynamic maze using deep Q-networks

Apart from the tabular versions of Q-learning, the utilisation of deep Q-networks was examined. A simple multilayer-perceptron approach with two hidden layers was implemented, with ReLU or PReLU activation functions. The Adam optimiser was used and the learning rate was set to 0.001. Multiple sizes for the hidden layers were used, specifically

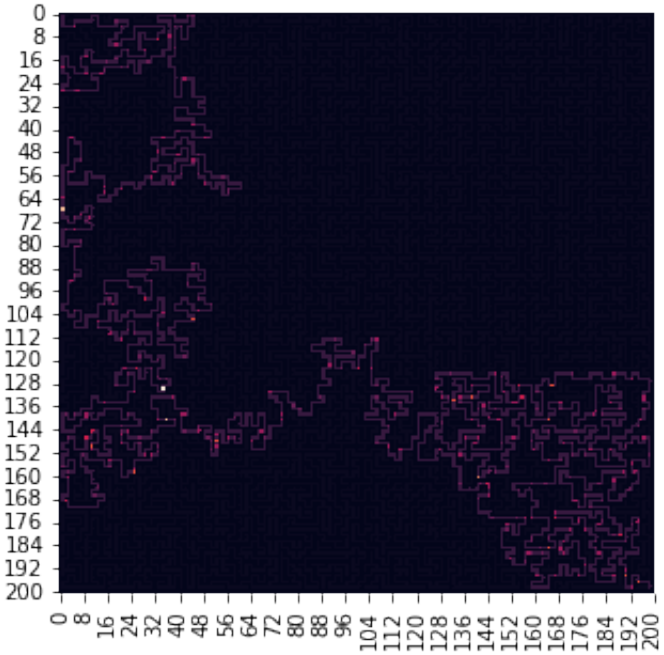

Fig. 7. The extracted trajectory of the agent in the case of a dynamic maze using tabular Q-learning. Brighter colours correspond to locations in the maze that the agent traversed multiple times. The brighter the colour the more time the agent spent in these locations.

configurations with size 20, 128, 256, 512 and 800 were used for both hidden layers. For the experience replay buffer the maximum memory varied from 200 to 1000 episodes in the conducted experiments. The sample size extracted from the buffer at every step of the training phase was varied between 48 and 500, while the batch size varied from 16 to 100. The target network update time was varied per a number of steps between 1 and 50. However, the learning process for the model was significantly slower than the tabular case of Q-learning examined in sections IV.A and IV.B. Training the deep Q-networks for about 20 hours, the agent had traversed about a quarter of the maze, whereas at the same time the approaches of tabular Q-learning consistently reached the (199,199) tile, while the Q-table for the static case of the maze had already converged to a solution to the problem. Therefore, the approach utilising deep Q-networks was not further examined and more work was put towards further optimising the tabular approaches. Nevertheless, the source code for the deep Q-networks used is available through the github link provided in section I, both for future experiments and reproducibility purposes.

V. Discussion

A. Proposed approach

Many obstacles were faced in the process of solving the maze. As mentioned in section IV, there were problems with a double Q-learning approach in the tabular case of Q-learning. Investigating the issue further led to the following observation: in the case that one of the two tables had not been updated yet upon initialising the two Q-tables, an action chosen from the updated table A to be used towards updating table A using the respective Q-value of the non-updated table B could lead to wrong updates for table A. While initially the problem does not seem significant as sufficient training should resolve the issue, when combined with the framework presented in Figure 5 it can significantly affect training time due to the use of a dynamic reward function. Therefore, a conclusion to draw from this process is that in the case that dynamic reward functions are used with tabular Q-learning, it would be best to utilise a double Q-learning approach with Q-tables that have been initialised based on a sufficiently trained Q-table derived from a single-table Q-learning approach first.

A second issue faced had to do with the learning rate for the tabular Q-learning approaches. Breaking down the problem into smaller mazes, e.g., the objective being to reach the (20,20) tile rather than the (199,199) one, it was observed that while having a constant learning rate would be sufficient for convergence in the case of relatively small mazes, that is not the case with larger ones. This has to do with the complexity of the combinatorial aspects of the maze due to its size. As the maze increases in size the amount of possible routes the agent can take increases significantly. Therefore, the agent may end up traversing a tile only once while having traversed another tile hundreds of times. This leads to a non-uniform distribution of updates for the Q-values and in turn, it is empirically evidenced that this leads to prohibitive slow convergence for the Q-table. Using the framework of Figure 5 however, and selecting the learning rate appropriately as described in sections IV.A and IV.B, notably accelerated the convergence of the Q-tables. Therefore, a conclusion to draw in this case is that a decreasing learning rate, and particularly a learning rate converging to zero, is imperative for the convergence of Q-tables for problems with large space of states.

Furthermore, it is worth to note that the tabular approach adopted in this work cannot provide a complete solution to the problem, as the space of states examined is only a subspace of the actual space of states due to the stochastic nature of the maze. In theory, deep Q-networks would be a more appropriate approach to consider, due to the flexibility they provide in terms of the space of states. By having the ability to receive vectors as input, the space of states of the current problem can be sufficiently represented as a 20-dimensional space with real coordinates. However, in practice the tabular Q-learning approach is much faster, with significantly fewer hyper-parameters to tune. Furthermore, combining the results from the Q-table derived for the static case of the maze with the strategy for the agent to remain in its position when fire is on the tile with the highest Q-value, consistently solves the maze in about 4,000 to 4,500 steps depending on the specific stochastic events that occur. Therefore, the static solution of the maze can be effectively extended in order to provide a solution to the dynamic one as well, and is currently the working solution of the proposed approach for traversing the maze successfully in the general case scenario. The results in Figure 7 correspond to such a solution.

B. Alternative approaches considered

Other approaches were considered towards solving the maze, primarily regarding the appropriate definition for the reward function. Initially, the idea of rewarding actions that get the agent geometrically closer to the (199,199) tile was considered. However, dead ends and shifts in the agent's direction that are necessary in order to reach the (199,199) tile do not allow the adoption of such an approach.

Furthermore, the idea of having two agents, one starting from tile (1,1) and the other starting from tile (199,199) was examined. In that scenario the objective would be for the two agents to meet as soon as possible. By sharing information about their locations at certain time steps, the two agents could be possibly trained towards a solution to the problem while reducing the area they would have to explore individually. Upon achieving a solution to this problem, the trajectory of the agent starting from tile (199,199) could be modified appropriately and the Q-values for its state-action pairs could be used to inform the Q-values of the agent starting from tile (1,1). However, while this is an interesting approach to the problem, it also omits its objectives and was not examined any further.

Another approach considered was training multiple agents starting from different locations of the maze. The motivation behind this idea is that it could be possible that having more agents exploring a smaller area of the maze would be faster than having one agent explore the whole maze. However, as in the case of communicating agents described above, this would omit the objectives of the problem.

C. Limitations of the proposed approach

While the proposed approach is sufficient towards solving the maze, it should be noted that there are certain limitations due to the assumptions made and the nature of the problem. Firstly, the proposed approach is based on the fact that the agent will always start from tile (1,1). If the agent was e.g., randomly placed in the maze and the objective was to reach the (199,199) tile, an extension of the proposed approach should be considered instead. Secondly, the specific structure of the maze allows for the framework presented in Figure 5 to provide sufficient results. In the case of mazes where dead ends corresponded to halls rather than single tiles, the proposed approach would work only up to a certain extend and more work would be necessary to formulate the problem accordingly. An example of a case where the proposed approach would not be sufficient is given on Figure 8.

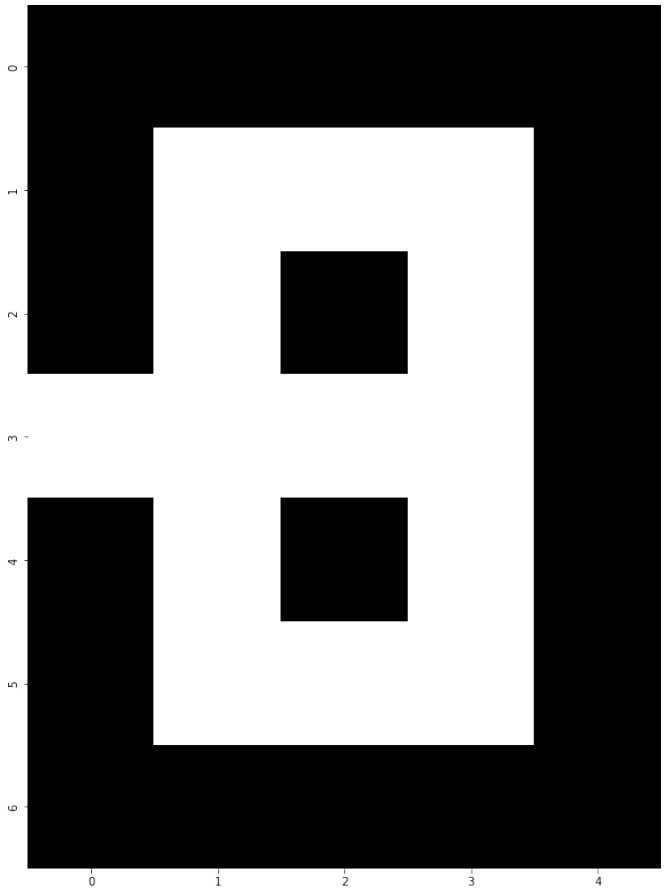

Fig. 8. An example of a hall. This hall is essentially a dead end. However it would not be possible to be captured sufficiently using the proposed approach. In that case, more work should be done towards extending the framework utilised in this work.

VI. Conclusion

This work demonstrated an approach based on tabular Q-learning towards solving a dynamic maze problem. With the aid of a dynamic reward function and a framework that utilises past information collected while traversing the maze, the problem can be solved efficiently. The challenges arising from the dynamic elements of the maze were discussed along with the performance of deep Q-networks towards solving the maze. Alternative approaches were presented that could lead to new intuitive solutions towards solving maze problems in the general case scenario, while the limitations of the proposed approach were highlighted in order to facilitate further work towards upgrading its generalisation capabilities.

References

[1] H. v. Hasselt, "Double Q-learning," Advances in Neural Information Processing Systems, vol. 23, 2010

[2] H. v. Hasselt, A. Guez and D. Silver, "Deep Reinforcement Learning with Double Q-learning," Proceedings of the AAAI conference on artificial intelligence, vol. 30, No. 1, 2016

VII. Appendix

A. Compiling and running the code

All software was run in Google Colab in order to make the code easily accessible and easy to run for anyone. The source code files accompanying this report are organised in a compressed folder, named "ktd1g20_COMP6247-20220527T132109Z-001". Once the folder has been decompressed it should be uploaded to Google Drive directly in the default space (My Drive). There are three subfolders named: "OutputFileGeneration", "QLearningDynamicMaze"

and ”QLearningStaticMaze”. In each subfoler there is an .ipynb file. There is no need for any modification in the code as long as the initial decompressed folder has been appropriately uploaded in the default space of a Google Drive account. The .ipynb files must then be opened and executed in order to run the code depending on what is desirable. Note that an authorization prompt will pop up upon executing the .ipynb files. Access needs to be granted in order for the code to run in all cases :

- The .ipynb file in ”OutputFileGeneration” will produce a .txt file that has the information of what the agent observes and the actions it takes in order to reach from the (1,1) tile to (199,199) while respecting the random fire. The code outputs the path together with the timing information and shows the trace of the agent in the environment. Furthermore, a heatmap is produced that captures the agent's trajectory. This is the working solution of the approach proposed in this work and should always solve the dynamic maze. This should take about 2:30 minutes in Google Colab and the number of steps needed, empirically, vary between 4,000 and 4,500. The extracted output .txt file can be found in the same folder as the corresponding .ipynb file upon execution.
- The .ipynb file in ”QLearningDynamicMaze” contains the training code for the agent using tabular Q-learning for the dynamic case of the maze. It should be noted that several files will be created after a certain point of executing the file in order to allow the utilisation of the proposed framework. Furthermore, training the agent to reach the (199,199) tile in training mode could take a significant amount of time. Once the agent has been sufficiently trained towards reaching the (199,199) tile in training mode, the approach adopted in section IV.B regarding the learning rate can be implemented, by retraining the agent using the so far extracted set of ”bad intersection moves” and the set of ”bad states” as explained in the code's comments.
- The .ipynb file in ”QLearningStaticMaze” contains the training code for the agent using tabular Q-learning for the static case of the maze. It should be noted that several files will be created after a certain point of executing the file in order to allow the utilisation of the proposed framework. Furthermore, training the agent to reach the (199,199) tile in training mode could take a significant amount of time. Once the agent has been sufficiently trained towards reaching the (199,199) tile in training mode, the approach adopted in section IV.A regarding the learning rate can be implemented, by retraining the agent using the so far extracted set of ”bad intersection moves” and the set of ”bad states” as explained in the code's comments.

B. Deep Q-networks code

As at the moment the deep Q-networks of this work do not provide a solution to the problem even in training mode, the relevant source code is not included in the file accompanying this report. However, as mentioned in section IV.C the source code for the deep Q-networks is available through the github link: https://github.com/DKT91?tab=repositories along with information on how to use them. The rest of the files presented in section VII.A are available through that link as well.