B143 Assessment Brief 2025

A game board with cars and trucks

AI-generated content may be incorrect.

**Project title:** Maze game with dynamic obstacle locations

**Course:** B143 -AI studio

**Name:** Mohammad Javadi Parvaneh

**Student ID:** GH1029707

GitHub Repository:

Table of Content:

Introduction…………………………………………………………………………...…………….

Methodology: ………………………………………………………………………………………

Algorithm: ……………………………………………………………………..…………………...

Appendix A- Full Source Code …..………………………………………………………………...

Experiments & Results: …………………………………………………...……………………….

Analysis: ……………………………………...……………………………………………………

Reflection Section……………………...………………………………………...…………………

Conclusion: ………………………...…………………………………………………...………….

Referencing:………………………………………………...……...………………………………

Introduction:

Maze Navigation:

In this part, I will explain the game, goal, and RL concept.

First and foremost, this game is inspired by Zen Garden paths or Sokoban puzzles.**🇯🇵**

I picked this game because it reminded me of my childhood. In addition, historically this game is used in educational and recreational material in different cultures. Following this, this game was also was popular in ancient labyrinths in Greek mythology, modern video games and robotics simulations. This game has a good and appropriate structure for learning algorithms. During the reinforcement learning process, agents make optimal decisions based on trial and error. In this process, an agent must learn to move from a starting position to a goal. Likewise, the path is often filled with obstacles or dead ends. The challenge is not only to reach the goal and it is beyond that. Following this, you have to do so efficiently and reliably through decision making under uncertainty.

There are many reasons why I chose maze navigation. For instance, due to the relevance of reinforcement learning and simplicity. This game has a clear definition of states, actions, and rewards, and all of these features are part of any RL problem. The most important thing, this game has a grid based layout and it will help to make it easy to visualize the agents learning progress over time. In my point of view, it would be so important for me and the result. This game meets all the criteria that are set by the assessment brief and is also suitable and effective option for applying and evaluating RL algorithms like Q-learning or Deep Q-Networks(DQN).

Methodology:

In this part, I will explain the Environment setup, state, actions, and reward design.

I first tried to design a simple 10x10 grid based environment. The agent will start in a fixed cell and must reach a designated goal cell. Based on the goal of our project, I should add complexity and variability to the environment. Following this, the agent has to adapt to a new situation in every run. In each episode, a new set of grid cells is randomly marked as an obstacle, but we will ensure that the start and goal positions remain clear. For implementing the all game we are using a NumPy array. Each cell holds a value and it represents empty space, a wall, the agent’s location, or the goal.

Following this, each state is defined by the agent’s current position on the grid, including the tuple (x,y). Based on this agent is supposed to have four possible actions: move up, move down, left, or right and that is all. Due to the agent’s movements, if the agent tries to move into a wall or outside the maze, then his move will be rejected and the agent will receive a penalty. It would be small and would be based on the rules that we define at the start of the game. Whenever the agent reaches the goal it gives a reward of +1 and the game will end. Each step taken incurs a small negative reward and that leads to shorter paths.

In terms of setup, I will use the Q-learning algorithm to train my agent.

Q-values are also stored in a table that is indexed by state action pairs. The agent selects actions using an ε-greedy policy, balancing exploration and exploitation. During this time, ε will decrease. The learning rate would be 0.1 and a γ (discount factor) would be 0.95. In addition, training performance will evaluated by tracking agent cumulative rewards and the number of steps that are so important to reach the goal across episodes.

Algorithm:

In this part, I will explain my RL technique and the process of implementation details.

The goal of the agent is to learn best optimal policy that leads to maximizing the cumulative reward over time by learning. Following this, each action would be different in each state of the maze.

For my project, I implement the Q-learning algorithms. It is well known and famous model free reinforcement learning method. The agent is using the table called Q-table and it has one update rule which I will mention later. The agent learns these values through repeated interaction with the environment. All values will be stored in a particular state.

Our rule is like the picture below:

A math equation with black text

AI-generated content may be incorrect.

A white background with black text

AI-generated content may be incorrect.

In my project, our game environment contains a 10x10 grid maze with dynamically changing obstacles. Before each episode, a new set of obstacle positions is generated at random. The most important thing is, our start and goal cells are constant. Following this, this changes will happen and agent cannot rely on memorizing a fixed path and must learn to generalize its policy. As I mentioned before, our actions also include four discrete movements: up, down, left, and right. The training was run for 5000 episodes. An ε factor greedy strategy was used to balance exploration. Finally, an ε will be set to 1.0 to encourage exploration and this amount is decreasing constantly to 0.01. Likewise, α as a learning rate will be set to 0.1 and the symbol of γ (discount factor) will be set to 0.95. In the beginning, the Q-table will fill with zero and it will constantly change by agent progress and traying. Such a change helps to be able to find increasingly efficient paths to the goal.

**Appendix A- Full Source Python Code:**

I will paste the complete python implementation of the maze navigation agent by using the Q- learning method. It included in Appendix A for reference.

#lets start with importing some libraries

import pygame

# we need it for showing the maze and agent visually

import numpy as np

# we need it for math and matrix like q-table

import random

# we need it for generating random numbers

from collections import deque

# we need it for storing past positions ( I will do that to avoid loops and repetitive movements)

# It is time to set maze and learning items:

maze\_size = 10

# we need maze with size of 10\*10

start = (0, 0)

# start point for agent is 0,0

goal = (9,9)

num\_obstacles = 15

#I think 15 obstacle is enough

#Let's define Q-learning parameters

alpha = 0.1

# alpha is learning rate

gamma = 0.95

#it is discount factor

epsilon = 1.0

#this is exploration rate

epsilon\_min = 0.01

#smallest possible exploration

epsilon\_decay = 0.999

episodes = 5000

#this is number of training rounds

max\_steps = 100

#max steps per episode

#let's define the actions

actions = {

0: (-1, 0), #for up

1: (1, 0), #for down

2: (0, -1), # for left

3: (0, 1) #for right

}

# let's define Q-table initialization:

#Following this, this table stores the learned values for each cell and action

q\_table = np.zeros((maze\_size, maze\_size, 4))

#Based on my professor the obstacles supposed to be different in each episod

#let's define the definition for creation of that:

def generate\_obstacles():

obstacles = set()

while len(obstacles) < num\_obstacles:

x, y = random.randint(0, maze\_size-1), random.randint(0, maze\_size-1)

if (x,y) !=start and (x, y) !=goal:

obstacles.add((x, y))

return obstacles

#we have to check if a position is valid or not. I mean that inside grid and not blocked.

def is\_valid(pos, obstacles):

x, y = pos

return 0 <= x < maze\_size and 0 <= y < maze\_size and pos not in obstacles

# We have to check the maze is solvable or not, so

#I use BFS to confirm there is a path from strat to goal

#Let's define the definition:

def path\_exists(start, goal, obstacles):

queue = deque([start])

visited = {start}

while queue:

x, y = queue.popleft()

if (x, y) == goal:

return True

for dx, dy in actions.values():

nx, ny = x + dx, y + dy

if is\_valid( (nx, ny), obstacles) and (nx, ny) not in visited:

visited.add((nx, ny))

queue.append((nx, ny))

return False

# There are many possibilities whenever agent wanted to move in the environment

# so, we have to change and predict his moves to reach the better and faster result

def step(pos, action, obstacles):

dx, dy = actions[action]

new\_pos = (pos[0] + dx, pos[1] + dy)

if not is\_valid(new\_pos, obstacles):

return pos, -1 # this would be for hiting the wall or obstacles and causes penalty

if new\_pos == goal:

return new\_pos,100 # if agent will reach the goal he will receive big reward

return new\_pos, -0.5 #I set the small penalty for his regular step

# let's train the agent using the Q-learning

for ep in range(episodes):

obs=generate\_obstacles()

while not path\_exists(start, goal, obs): # for make sure about maze is solvable

obs=generate\_obstacles()

pos = start

for \_ in range(max\_steps):

#Decide whether to explore or use best known move

if np.random.rand() < epsilon:

action = random.choice(list(actions.keys()))

else:

action = np.argmax(q\_table[pos[0], pos[1]])

next\_pos, reward = step(pos,action, obs)

next\_max = np.max(q\_table[next\_pos[0], next\_pos[1]])

#so, let's update Q-table using the Q-learning formula

q\_table[pos[0], pos[1], action] += alpha \* (

reward +gamma \* next\_max - q\_table[pos[0], pos[1], action])

pos = next\_pos

if pos == goal:

break

#another important thing would be slowly reduce exploration over time

epsilon = max(epsilon\_min, epsilon \* epsilon\_decay)

#As we have learned during the class, we have to define pygame setup for visual display:

pygame.init()

cell\_size = 600 // maze\_size #this is the size of each cell on the screen

#so let's set the windows size

screen = pygame.display.set\_mode((600, 600))

pygame.display.set\_caption("smart agent (no loops)")

clock = pygame.time.Clock()

font = pygame.font.SysFont(None, 36)

#Let's draw maze, obstacles, agent, and goal:

def draw(maze, agent, message=""):

#I want to have ligh green background because of the real maze game appearance,so:

screen.fill((230, 255, 230))

for x in range(maze\_size):

for y in range(maze\_size):

rect = pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)

if (x, y) in maze:

pygame.draw.rect(screen, (0, 100, 0), rect) #obstacles are supposed to be dark green

elif (x, y) ==goal:

pygame.draw.rect(screen, (255, 0, 0), rect) #goal color is supposed to be red

elif (x, y) == agent:

pygame.draw.rect(screen, (0, 0, 255), rect) #For the color of agent and it would be blue

else:

pygame.draw.rect(screen, (200, 255, 200), rect, 1) #For grid lines

if message:

text = font.render(message, True, (0, 128, 0))

screen.blit(text, (10, 10))

pygame.display.flip()

# I have to limit agent's movements whenever it runs through Maze

def run\_agent(obstacles):

pos = start

prev\_positions = deque(maxlen=6) #For storing recent positions to detect cycles

for \_ in range(max\_steps):

draw(obstacles, pos)

clock.tick(5)

#choosing best action from Q-table

q\_values = q\_table[pos[0], pos[1]]

action\_order = list(np.argsort(q\_values)[::-1])

#For trying the best actions first to have a better performance.

moved = False

for a in action\_order:

dx, dy = actions[a]

new\_pos = (pos[0] + dx, pos[1] + dy)

if not is\_valid(new\_pos, obstacles):

continue

if new\_pos in prev\_positions:

continue #For avoiding the loops

prev\_positions.append(pos)

pos = new\_pos

moved = True

break

#For handling the window close event we have

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit(); exit()

#For time that agent is stuck or looping

if not moved:

draw(obstacles, pos, "Agent stuck or looping")

pygame.time.wait(1500)

return False

#For the time that agent reached the goal

if pos == goal:

draw(obstacles, pos, "Agent reached the goal!!")

pygame.time.wait(1500)

return True

return False

#For showing 5 successful runs visually

successes = 0

while successes < 5:

obs = generate\_obstacles()

if path\_exists(start, goal, obs):

if run\_agent(obs):

successes += 1

pygame.quit()

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a computer game

AI-generated content may be incorrect.

Experiments & Results

In this part, we will look at the graphs, tables, evaluation metrics, and other findings.

Now, it is time to evaluate the effectiveness of the reinforcement learning based on our maze navigation task. Furthermore, a series of experiments were conducted using that 10x10 grid maze with static obstacles. These static include static obstacles a fixed start point and one goal state. All of these conditions have to be considered to reach appropriate results. I will use NumPy, and Q- learning to train the agent.

To evaluate the effectiveness of the reinforcement learning agent in the maze navigation task, a series of experiments were conducted using a fixed 10x10 grid maze with randomized obstacles in each episode, making the task dynamic and more realistic I think. Based on our agreement, despite the changing layouts, the agent was still able to improve over time. That improvement, demonstrating its ability to generalize. The environment was implemented using NumPy, and Q-learning was applied to train the agent. The training phase was run over 5000 episodes, with a maximum of 100 steps per episode. As I mentioned before α as a learning rate was set to 0.1, and the symbol of γ (discount factor) will be set to 0.95. The exploration parameter will be defined as epsilon and it will decrease linearly from 1.0 to 0.01 over time. As I mentioned before the Q-table was initialized to zero and it will update during the time based on Q-learning update rules.

Evaluation Metrics

There are three primary metrics were used to evaluate the learning progress from the agent side:

**1. Cumulative reward per episode:** It shows how well the agent is maximizing its long term rewards.

**2. Number of steps per episode:** I measure efficiency in reaching the goal.

**3-Success rate:** Measuring whether the agent was able to reach the goal within the episode limit.

**Results:**

By looking the exact look at the table below you can see how my agent’s learning over time will change.

|  |  |  |  |
| --- | --- | --- | --- |
| Episode Range | Average Reward | Average steps to Goal | Success Rate |
| 1-500 | -0.3 | 95 steps | 8% |
| 2001-2500 | 0.4 | 42 steps | 73% |
| 4501-5000 | 0.9 | 22 steps | 92% |

A graph with a line

AI-generated content may be incorrect.

By looking the exact look at the Total reward per episode column:

We can see an upward trend and it indicate an improved performance. At the beginning, early episodes had low or negative rewards and after 1.500 episodes, our reward increased steadily as the agent learned more and more paths.

By looking the exact look at the Total reward per episode column:

We can see a clear noticeable decrease during this time. That trend will confirm that the agent was able to reach the goal with fewer steps in comparison with the past. That means the training process progressed and it worded.

By looking the exact look at the success rate column:

We can also see the agent successfully improve his goal reaching ability, it will rise from near 0% to over 90% in the final stages of training and that is a good sign.

To sum up, based on the aforementioned compelling grounds, the agent successfully learned to navigate the maze environment by experiencing trial and error using the Q-learning method. We can say that we have effective policy learning over time and the results are also approving this.

Analysis:

According to the previous table, the agent shows many signs of learning. Which I mentioned before. During this course of training, the agent reached higher cumulative rewards, reduced steps toward the goal, and significantly increased success rate. All of these aspects are so important and have meaningful progress from our point of view.

Strengths of Q-learning:

**1-Simplicity & Efficiency:** Q-learning is easy to implement and it is light in terms of computationally. It will need not deep learning and it will be able to learn all policies during a few thousand episodes.

**2-Clear Improvement:** Q-learning can find the appropriate and efficient paths. Our observation in terms of the agent’s success shows that the success rate has increased steadily over our specific training. No matter whether our maze includes fixed obstacles or paths, our agent becoming better and better.

**3-Interpretable Q-table:** the Q-table gives direct insight into the learned behavior, that feature is useful for analysis and Debugging. This feature is unlike deep learning based methods and it is so important.

4**-The agent can now** generalize better across unseen and different maze layouts thanks to randomized obstacles.

Weaknesses of Q-learning:

**1-Exploration dependency:** The most important thing is the epsilon schedule because the learning rate is highly sensitive to the choice of the epsilon schedule. Following this, if the epsilon amount decreases too quickly, our agent will fail to explore the maze sufficiently and it will get stuck in suboptimal paths.

**2- Poor and weak Generalization:** Q-learning also has some key limitations in big or dynamic environments. That means, our agent was trained on a specific maze configuration and if the layout changes, the agent must re-learn from scratch.

**3-Inefficiency in Large state spaces:** The function of Q-learning will become slower in terms of coverage. That means, even a 10x10 grid with multiple obstacles is assuming large and this is not enough for our processes.

4**-The dynamic nature of the environmen**t makes convergence slower, but I think it reflects a more realistic challenge.

Possible improvements:

**1-Dynamic or Randomized mazes:** We can use a function approximation method, for example, a neural network instead of a Q-table to have a better outcome. Following this, we could train the agent to generate better. Introducing multiple mazes with varying layouts are also could help to train the agent better.

**2-Use of Deep Q-Network (DQN):** For improving scalability and allowing the learning in larger scale or more complex mazes we can switch from tabular Q-learning to DQN.

**3-Reward shaping:** We can encourage the agent to learn and understand shorter and faster paths more quickly by adding small negative rewards for each step and stronger positive rewards due to faster completion.

**4-Training Enhancements:** We can use some techniques like double Q-learning or prioritized experience replay, which leads our agent to faster convergence and better performance.

Reflection

Working on this project was a great experience for me because it helped me to increase my understanding learning and, especially how Q-learning adapts by using trial and error in one environment. During our experiment, we have noticed that small changes in the learning rate or discount factor impact our agent’s performance. Likewise, it was so interesting to me to see the agent’s learning process over episodes, and it helped me to appreciate the importance of reward shaping and state representation. This project was so important for me because, simultaneously, I strengthened my coding and debugging skills and interpreted learning patterns from data.

Conclusion:

To sum up, I have tried to develop a reinforcement learning agent that learns to navigate a maze environment using the Q-learning algorithms. The agent improved its performance over time through trial and error by using the training loop. During that process, we designed a suitable environment, we defined some rules like state, action, reward, and penalty structure.

By having an exact look at the results, we can see a clear and notable learning pattern, which means the agent gradually improves his ability during the time. That means he reaches the goal more efficiently and consistently. During the experiment, we confirmed our assumption which was even a simple tabular Q-learning approach can be fruitful in solving deterministic navigation tasks in maze.

On the other side, we saw some limitations of Q-learning as well. As I mentioned before, this method does not fit well in larger and more dynamic environments and multiple layouts also make the situation harder. All these weaknesses would be some sparks for some potential directions and future work.

Future Directions:

**Implement Deep Q-Network (DQN):** we should consider this to handle larger and bigger state spaces and also generalization across multiple maze configurations.

**Explore Transfer Learning:** We should think about our trained agents and the way in which they adapt themselves to new environments with minimal retraining and so on.

**Add Dynamic Elements:** We should maybe like to move obstacles or time constraints and make our tasks more complex and realistic.

**Experiment with Alternative Algorithms:** We could think about using algorithms such as SARSA, Double Q-learning, or Actor-Critic methods to compare performance and stability.

To sum up, based on the aforementioned content that I provided, my project is valuable hands-on experience in applying RL techniques. I have mentioned some valuable insights about algorithms like Q-learning. Such as the strengths and limitations of this method. In my point of view, I provided a strong foundation for further exploration of more advanced approaches in reinforcement learning.

References:

1. 1-Quokka. (n.d.). Quokka Learning wood maze games for kids ages 1-3- ver.2 improved. From <https://quokka.com/products/quokka-learning-wood-maze-games-for-kids-ages-1-3-ver-2-improved>
2. 2-Sutton, R.S. and Barto, A.G. (2018).Reinforcement Learning: An Introduction. 2nd ed. Cambridge,MA:MIT Press.
3. 3-Watkins, C.J.C.H. and Dayan, P.(1992). Q-learning. Machine Learning, 8(3-4),pp.279-292.https://doi.org/10.1007/BF00992698.
4. 4-Mnih, V. et al.(2015). Human-level control through deep reinforcement learning.Nature,518(7540),pp.529-533. <https://doi.org/10.1038/nature14236>
5. 5-Pygame Community (2024).Pygame Documentation.https://www.pygame.org/docs
6. 6-NumPy Developers (2024).NumPy Documentation. <https://numpy.org/doc>
7. 7-Kaelbling, L.P., Littman, M.L and Moore,A.W.(1996).Reinforcement learning: A survey.Journal of Artificial Intelligence Research, 4,pp.237-285.https://doi.org/10.1613/jair.301
8. 8-Van Hasselt, H., Guez, A.and Silver, D.(2016).Deep Reinforcement Learning with Double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence,30(1).Available at: https:https://ojs.aaai.org/index.php/AAAI/article/view/10295
9. 9-Schaul, T., Quan, J., Antonoglou, I.and Silver,D.(2016).Prioritized experience replay. In International conference on learning representations(ICLR).Available at:https://at:https://arxiv.org/abs/1511.05952
10. 10-Zhang,S.,Vinyals, O.,Munos,R.and Bengio,S.(2018).A Study on Overfitting in Deep reinforcement learning.In proceedings of the 35th international conference on machine learning(ICML). Available at:https://arxiv.org/abs/1804.06893
11. 11-Silver, D.et al.(2016).Mastering the game of Go with deep neural networks and tree search.Nature,529(7587),pp.484-489.https://doi.org/10.1038/nature16961
12. 12.Rummery,G.A.and Niranjan,M.(1994). On\_line Q-learning using connectionist system.Technical Report CUED/F-INFENG/TR166.cambridge university engineering department.Available at <https://www.repository.com.ac.uk/handle/1810/271010>.
13. 13-Konda,V.R. and Tsitsiklis,J.N.(2000).Actor-critic Algorithms.In: Advances in Neural information processing systems(NIPS 1999), pp.1008-1014.available at: https://papers.nips.cc/paper\_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-paper.pdf
14. 14.Taylor,M.E. and stone, P.(2009).Transfer learning for reinforcement learning domains:A survey.Journal of machine learning research,10,pp.1633-1685.available at:https://www.jmlr.org/papers/volume10/taylor09a/taylor09a.pdf
15. 15.Ng,A.Y.,Harada,D.and Russell,S.(1999).policy invariance under reweard transformations:theory and application to reward shaping.In ICML 1999: proceedings of the sixteenth international conference on machine learning,pp.278-287.available at https://www.cs.cmu.edu/~bziebart/Ng\_Harada\_Russell.pdf