Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
17/2 (2024), 145-158. DOI: http://dx.doi.org/10.21609/jiki.v17i2.1236

Application of Q-learning Method for Disaster Evacuation Route Design Case Study: Digital Center Building UNNES

Hanan Iqbal Alrahma, Anan Nugroho*, Ahmad Fashiha Hastawan[3], Ulfah Mediaty Arief[4]

[1,2,3,4]Faculty of Engineering, Universitas Negeri Semarang, Gunungpati, Semarang, 50229, Indonesia

E-mail: [1]hananiqbal99@students.unnes.ac.id, [2]anannugroho@mail.unnes.ac.id,
[3]ahmad.fashiha@mail.unnes.ac.id, [4]ulfahmediatyarief@mail.unnes.ac.id

Abstract

The Digital Center (DC) building at UNNES is a new building on the campus that currently lacks evacuation routes. Therefore, it is necessary to create an evacuation route plan in accordance with the Minister of Health Regulation Number 48 of 2016. Creating a manual evacuation route plan can be inefficient and prone to errors, especially for large buildings with complex interiors. To address this issue, learning techniques such as reinforcement learning (RL) are being used. In this study, Q-learning will be utilized to find the shortest path to the exit doors from 11 rooms on the first floor of the DC building. The study makes use of the floor plan data of the DC building, information about the location of the exit doors, and the distance required to reach them. The results of the study demonstrate that Q-learning successfully identifies the shortest evacuation routes for the first-floor DC building. The routes generated by Q-learning are identical to the manually created shortest paths. Even when additional obstacles are introduced into the environment, Q-learning is still able to find the shortest routes. On average, the number of episodes required for convergence in both environments is less than 1000 episodes, and the average computation time needed for both environments is 0.54 seconds and 0.76 seconds, respectively.

Keywords: *evacuation route, reinforcement learning, Q-learning*

1. Introduction

The challenge of evacuation within a building, aside from the condition of the building's extensive and complex layout, lies in the lack of knowledge about internal connectivity within the structure. In such situations, individuals inside the building may not be aware of suitable evacuation routes, especially when facing unexpected events such as fires, tsunamis, or hurricanes. It is crucial to swiftly move occupants to safe areas to minimize potential damage and effectively manage emergency situations [1]. Therefore, planning navigation for optimal evacuation routes can reduce the likelihood of property damage, control congestion during emergencies, and shorten evacuation times to ensure that inhabitants can be safely and efficiently evacuated, ultimately saving more lives [2]–[4].

According to the Minister of Health Regulation (Permenkes) Number 48 of 2016 regarding Occupational Safety and Health Standards in Offices, every building is mandated to have a designated area used as an assembly point and an evacuation route diagram. The evacuation route diagram is intended to show where occupants should gather in case of an emergency and are instructed to evacuate [5]. The Digital Center (DC) building at UNNES is a building that can be categorized as an office building, constructed in 2019 and completed in 2021. The DC building is located on the west campus of Semarang State University, Sekaran, Gunungpati, Semarang City. Based on observations, the DC building has dimensions of 50x20 meters and consists of 4 floors, with a total of 44 rooms. As it is a relatively new building on the campus area, the DC building does not currently have evacuation routes. Therefore, it is necessary to create an evacuation route map for the DC building to comply with the regulations stated in the Minister of Health Regulation Number 48 of 2016.

Planning evacuation routes within a building can be done manually by calculating the shortest distance from the starting point to the endpoint. However, this method is less efficient, involves a high workload, and has a high potential for errors [6]. Therefore, automatic pathfinding algorithms like A-star and Dijkstra are employed. With the advancement of technology, automatic pathfinding is not limited to A-star and Dijkstra algorithms [7]. Many other algorithms are utilized, including genetic algorithms (GA), ant colony optimization, neural networks, and machine learning techniques such as reinforcement learning [2], [3], [7].

Several studies demonstrate evacuation route planning using various methods. A study conducted by Wu, Kang, and Wang aims to identify methods for simulating the evacuation process indoors. The study proposes a Cellular Automaton Crowd Route Choice (CACR) model under different conditions such as fire or sound warning systems. Experiments were conducted in a gymnasium with four exit doors for the stadium evacuation scenario. The results indicate that the CACR model achieved a total evacuation time of 3 minutes and 54 seconds [8].

In 2019, Peng et al. conducted research on evacuation route planning for a building by implementing a neural network, specifically the Back-propagation Neural Network. The study aimed to plan dynamic evacuation routes for a public building with 15 floors. The model was tested under four different conditions: during office hours with 36 people in the main building and 1045 people in the additional building, during business hours with 9 people in the main building and 1420 people in the additional building, during nighttime with 272 people in the main building and 30 people in the additional building, and during holidays with 105 people in the main building and 750 people in the additional building. The test results indicated that during office hours, the average evacuation time was 53.50 seconds, during business hours it was 49.47 seconds, during nighttime it was 121.24 seconds, and during holidays it was 57.63 seconds [3].

In 2021, Xue et al. conducted a study to develop an automatic evacuation guidance system in buildings by proposing the Combined Action Space Deep Q-Network (CA-DQN) method. Experimental results demonstrated that the proposed method achieved the fastest evacuation time, which was 31.65 seconds. In comparison, the Dynamic Shortest Path method required 32.18 seconds, and evacuation using static signs took 41.35 seconds [6].

The research conducted by Wu et al. in 2022, titled 'Evacuation Optimization of a Typical Multi-exit Subway Station: Overall partition and local railing, focuses on optimizing the evacuation of a subway station with multiple exit points by utilizing room partitions and local railings. The evacuation was simulated using Pathfinder, an emergency exit simulator based on an agent-based model (ABM). The study's results indicate that the use of local railings can balance the density distribution at different exit points, enabling evacuation to be completed in less than 200 seconds [9].

In 2022, Xu et al. utilized Q-learning for indoor emergency route planning. The research results indicated that the enhanced Q-learning approach could achieve the goal in 42 steps, converge after 500 iterations, and had a computational time of 13,738 seconds. These outcomes were superior to the State-Action-Reward-State-Action (SARSA) algorithm, which reached the goal in 102 steps, showed no convergence after 5000 iterations, and had a computational time of 164.86 seconds. Additionally, the classical Q-learning, when tested, achieved the goal in 42 steps, converged after approximately 2500 iterations, and had a computational time of 68,692 seconds [4].

Based on the research above, evacuation route planning has been conducted using various methods, some of which have employed reinforcement learning such as Q-learning and DQN. In the study by Xu et al, it has been demonstrated that Q-learning yields favorable results for designing evacuation routes within a building. Therefore, this method has the potential to be applied in designing evacuation routes for the DC UNNES building, which currently lacks an evacuation route layout.

2. Related Works

In addition to being used for pathfinding, Q-learning has also been applied in various other fields, as demonstrated in several studies. In the research conducted by Ardiansyah in 2017, Q-learning, combined with backpropagation, was employed to play the game Flappy Bird. The Q-learning with backpropagation required an average training time of 9 minutes and 1 second, while classical Q-learning took 120 minutes. Therefore, Q-learning with backpropagation was 92% faster than classical Q-learning with similar performance [14].

Furthermore, in a study by Low, Ong, and

Cheah in 2019, improved Q-learning was utilized to create optimal routes for a mobile robot. The research results indicated that Q-learning in simulation covered 22 units, and in real-world experiments, it covered 22.37 units with a deviation percentage of 1.68%. Additionally, the Improved Q-learning-Flower Pollination Algorithm (IQ-FPA) in simulation covered 24 units, and in the real world, it covered 23.57 units, with a deviation percentage of 1.79%. Improved Decentralized Q-learning (IDQ) in simulation covered a distance of 26 units, and in the real world, it covered a distance of 26.23 units, with a deviation percentage of 0.88% [10].

Li & Li conducted research using enhanced Q-learning, namely adaptive exploration Q-learning (AEQ), to solve path planning problems in an unknown environment. In scenario 1, both AEQ and Q-learning achieved the goal in 22 units, with an average computation time of 13.98 seconds for Q-learning and 13.08 seconds for AEQ. Meanwhile, SARSA reached the goal in 24 units, with a computation time of 13.86 seconds. In scenario 2, AEQ, Q-learning, and SARSA all required 23 units to reach the goal. The computation time for Q-learning in scenario 2 was 14.20 seconds, SARSA was 13.72 seconds, and AEQ was 12.53 seconds [11].

In a study conducted by Maoudj and Hentout in 2020, Q-learning was used for path planning in a mobile robot in various environments. In an environment with 8 obstacles, Q-learning covered 28.93 units with a computation time of 4.06 seconds. In an environment with 9 obstacles, Q-learning covered 30.67 units with a computation time of 4.04 seconds. In an environment with 10 obstacles, Q-learning covered a distance of 30 units with a computation time of 3.64 seconds [12].

The study by Zhang et al tested classical Q-learning and modified Q-learning methods, namely self-adaptive reinforcement-exploration Q-Learning (SARE-Q) and self-adaptive Q-Learning (SA-Q), on the OpenAI Gym grid environment. In a 20x20 grid environment, Q-learning required an average operation time of 2.047 seconds and an average number of steps of 23.38. SA-Q method required an average computation time of 1.739 seconds and an average number of steps of 24.44. Meanwhile, the SARE-Q method required an average computation time of 1.995 seconds and an average number of steps of 23.16. In a 10x10 grid environment, Q-learning required an average operation time of 1.43 seconds and an average number of steps of 24.24. SA-Q method required an average computation time of 1.034 seconds and an average number of steps of 24.16. The SARE-Q method required an average computation time of 1.147 seconds and an average number of steps of 24.04 seconds [13].

These studies collectively demonstrate the performance of Q-learning in solving various cases. Reinforcement learning is suitable for scenarios that lack a dataset or have limited data, such as evacuation simulation and pathfinding in unknown environments [15]. Therefore, Q-learning is suitable for designing disaster evacuation routes in the DC building. In this study, classical Q-learning was used, and its learning rate and reward system parameters were experimentally modified with reference to the research by [4].

Another consideration for using Q-learning is the current limitations of computational resources. Therefore, this research will be conducted in stages, starting with methods that require low computational resources and then proceeding to methods that require higher computational resources such as DQN. This way, we can make computation more efficient if more conventional methods can be optimal in handling pathfinding from an environment. The gradual implementation of Q-learning is also adopted by a number of studies [4], [10]–[13]. This is an advantage for several users who have limited computing machines.

3. Methodology

3.1. Environment Modelling

The environment used is the first floor of the UNNES Digital Center (DC) building, which was first represented in a floor plan with a scale of 1:100. This study only utilizes the first floor of the DC building because floors 2 to 4 have similar room placements and furniture arrangements as the first floor, both in terms of the number of rooms and layout. Thus, the mapping results on the first floor can be easily duplicated on floors 2 to 4. The first floor is prioritized for simulation in this research because it has the highest pedestrian traffic density and direct access to the building's entrance and exit doors. Figure 1 shows the floor plan of the DC building's first floor, and Table 1 presents the names and sizes of the rooms.

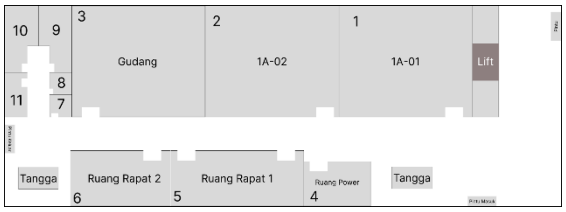


Figure 1. First floor plan

Table 1. Name and room size of DC building first floor

No.	Room Names	Size
1	Room 1A	12x10 meters
2	Room 1B	12x10 meters
3	Room 1C	12x10 meters
4	Power room	6x4 meters
5	Meeting room 1	12x5 meters
6	Meeting room 2	9x5 meters
7	Disabled toilet	2x2 meters
8	Cleansing room	2x2 meters
9	Toilet	6x3 meters
10	Pantry	5x2 meters

Next, the grid version of the environment is created using the Python programming language, utilizing the Tkinter library. The grid version of the environment is shown in Figure 2.

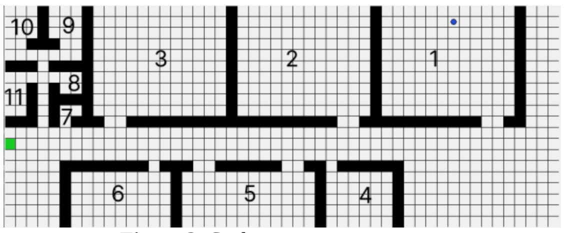


Figure 2. Grid version environment

The grid environment has dimensions of 50x20 grids. In the grid environment, the elevator is ignored as the agent will not pass through it. Each grid in the environment represents 1 meter in the real-world condition. This grid unit is also used as a unit of measurement for the number of steps the agent takes on the reference path as in Table 5, and the Q-learning path as in Table 6, etc. The total number of grids represents the number of states that the agent can traverse, which is 1000 states. The agent is depicted as a blue circle that can move within each room. The initial coordinates of the agent in each room can be seen in Table 5. The agent has 4 actions, namely moving up, right, down, and left. The goal is represented by a green square located at coordinates (12,0), which is the exit door of the building that the agent must reach, and it provides a reward of +10. White squares represent areas that the agent can pass through and have a reward of -1. On the other hand, black squares represent obstacles, and if the agent passes through them, it will receive a reward of -10.

3.2. Q-learning Implementation

Q-learning [16] is a model in RL (Reinforcement Learning). Q-learning belongs to the model-free and off-policy RL category. In off-policy RL, the agent attempts to build an optimal policy by directly interacting with the environment. In Q-learning, the agent uses a trial and error approach, wherein it repeatedly solves the problem using various approaches and continuously updates the policy as it learns about the environment [17].

In Q-learning, there is a Q-value that estimates how much additional reward can be obtained through all the remaining steps in the current episode if the agent is in state (s) and takes action (a). To calculate the Q-value, Formulation 1 is used.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \overset{max}{a} Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1)$$

In Equation 1, there is a learning rate (α) that usually takes values between 0 and 1. The learning rate parameter (α) signifies the rate of change from the old Q-value to be replaced by the new Q-value [14], [18]. A smaller learning rate implies a slower change in Q-value, indicating that the agent is cautious in updating Q-values. A low learning rate value (approaching 0) will slow down convergence, but in certain cases, it can prevent algorithm instability due to rapid Q-value changes. Conversely, if the learning rate is too large (approaching 1), the learning process will be faster, and convergence will occur more rapidly. However, this may lead to instability due to excessively fast Q-value changes [19].

The discount factor parameter (γ) is used to ensure that the rewards received by the agent remain bounded [14], [17]. The discount factor also influences the rewards obtained by the agent. If the discount factor is small (approaching 0), the agent will prioritize short-term rewards, whereas if the discount factor is high (approaching 1), the agent will prioritize long-term rewards [20].

The learning rate, discount factor, epsilon, and reward parameters are obtained as in the study by [4]. The determination of the values for learning rate, discount factor, epsilon, and reward is the result of experimental customization, ensuring that the Q-learning used is optimal for automatic pathfinding in the DC building. The parameter comparison can be seen in tables 2 to 4. The experimentation to determine the parameter values for learning rate, epsilon, and reward was conducted only in one room, namely room 1A or at coordinates (34,1).

Table 2. Experimental study of learning rate (α)

No.	Learning rate (α)	CPU Time (s)	Step (grid)	Episode
1	0.01	-	-	-
2	0.05	-	-	-
3	0.1	1.53	63	1042

Table 3. Experimental study of epsilon (ε)

No.	Epsilon (ε)	CPU Time (s)	Step (grid)	Episode
1	0.1	1.53	63	1042
2	0.5	7.9	63	-

No.	Epsilon (ε)	CPU Time (s)	Step (grid)	Episode
3	0.9	8.6	63	-

Table 4. Experimental study of reward

No.	Reward	CPU Time (s)	Step (grid)	Episode
1	+1, reach target location. -1, reach obstacle position. 0, reach other position	-	-	-
2	+10, reach target location. -10, reach obstacle location. -1, reach other location	1.53	63	1042

The Q-value calculated using Equation 1 is stored in the Q-table. The Q-table contains rows for each possible state and columns for each possible action. The optimal Q-table contains values that allow the agent to choose the best action in each possible state, thus providing an optimal path for the agent to achieve the highest reward. The Q-table represents the agent's policy for acting in the current environment. The Q-learning flowchart used in this case can be seen in Figure 3.

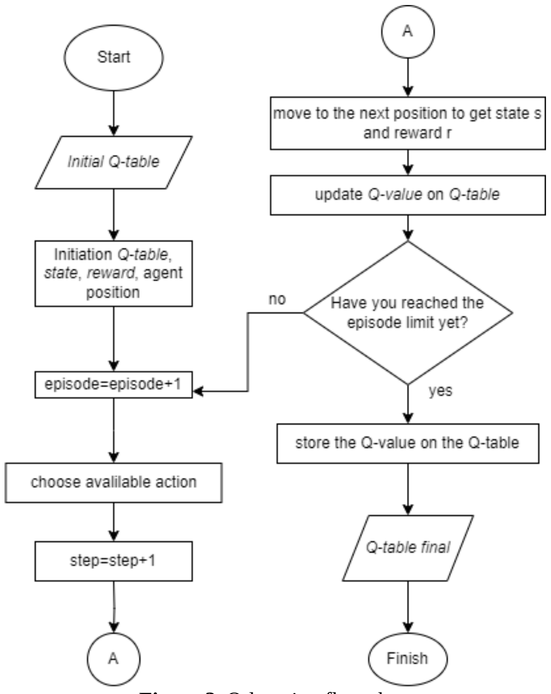

Start
Initial Q-table
Initiation Q-table, state, reward, agent position
episode=episode+1
choose avalilable action
step=step+1
A
A
move to the next position to get state s and reward r
update Q-value on Q-table
Have you reached the episode limit yet?
no
yes
store the Q-value on the Q-table
Q-table final
Finish

Figure 3. Q-learning flow chart

3.3. Performance Testing

The performance testing aims to analyze the ability of the Q-learning algorithm in planning evacuation routes. The system will be tested using a grid environment that resembles the first floor of the DC building, which consists of 11 rooms. The agent will be placed in each room with predetermined coordinates, as shown in Figure 4, where the agent is placed in room 1A with coordinates (40,1), and then training is conducted. After each training session is completed, the agent will be moved to the next room for a new training session, and so on until the agent occupies all the rooms. Table 5 shows the names of the rooms, their initial coordinate points, and the reference path lengths. The reference path is a route obtained through observation in the DC building and manually annotated which will be used as a comparison for the paths created by the Q-learning agent. The agent is required to reach the goal without colliding with any obstacles present.

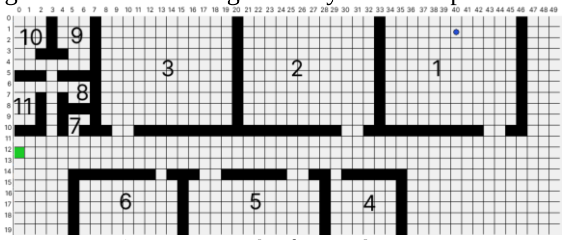


Figure 4. Example of agent placement

Table 5. Room names, initial coordinates, and references path lengths

No	Room Names	Initial Coordinates	References Path Length (grid)
1	1A	(34,1)	63
2	1B	(22,1)	49
3	1C	(10,1)	20
4	Power room	(34,17)	41
5	Meeting room 1	(27,19)	32
6	Meeting room 2	(6,19)	24
7	Disabled toilet	(7,9)	9
8	Cleansing room	(7,7)	13
9	Women's restroom	(7,1)	17
10	Men's restroom	(0,0)	18
11	Pantry	(0,6)	12

After the initial position of the agent is determined, the training process is conducted for a maximum of 3000 episodes using the following parameters: learning rate $\alpha = 0.1$, discount factor $\gamma = 0.9$, and exploration probability (ε) = 0.1, based on the experimental results presented in Tables 2, 3, and 4. The use of $\varepsilon = 0.1$ is intended to make the agent more inclined towards exploitation, meaning the agent is more likely to choose actions with the highest Q values.

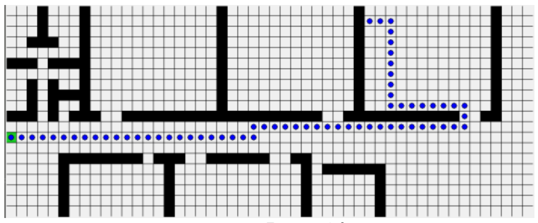
After the training process is completed, the evacuation routes created by the agent will be evaluated to assess the performance of the Q-learning algorithm in generating evacuation paths in the UNNES DC building. The paths created by the agent, will be measured in terms of the number of steps taken and then compared with the reference path lengths in Table 5. If the number of steps generated by the agent is the same as than the number of steps in Table 5, the results are considered acceptable. Furthermore, the path results generated by Q-learning will also be compared with another method, namely SARSA. SARSA is chosen as it represents a basic reinforcement learning method, similar to Q-learning, but they belong to different types – Q-learning is categorized as off-policy RL, while SARSA is considered on-policy RL. SARSA is frequently employed as a benchmark in prior research studies. To calculate the accuray of the paths generated by the Q-learning and SARSA agents, formula (2) is used. The reference used to calculate accuracy in this study is the reference path.

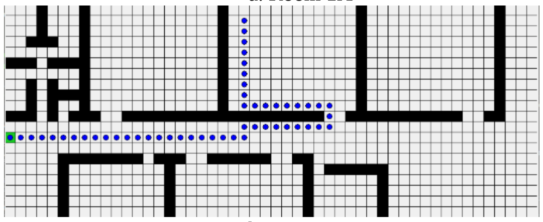
$$1 - \left(\frac{|agent\ path - references\ path|}{references\ path\ length}\right) \times 100\% \qquad (2)$$

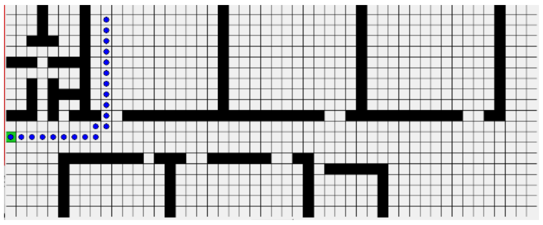
4. Result

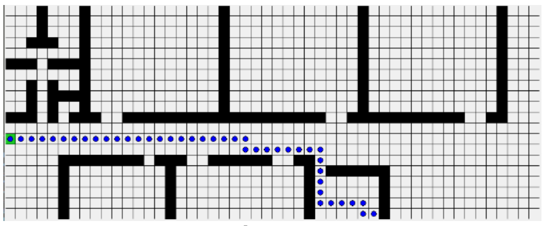
4.1. Comparison Q-learning with the reference path

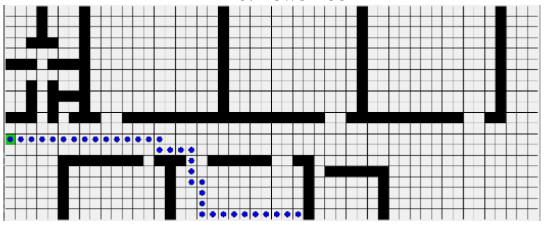
The simulation is conducted by placing the agent at the initial coordinates as listed in Table 5, in each room alternately. The simulation is implemented using Python with the help of the libraries tkinter, matplotlib, and numpy. The simulation is run on a laptop with the following specifications: Core i5 8250U 1.6 GHz processor, 12 GB RAM, and Windows 11 64-bit operating system. The shortest path generated by the Q-learning agent in the obstacle-free environment can be seen in Figure 5 a until k. The Q-learning agent successfully constructs a path from the predetermined initial coordinates to the goal without traversing the existing obstacles.

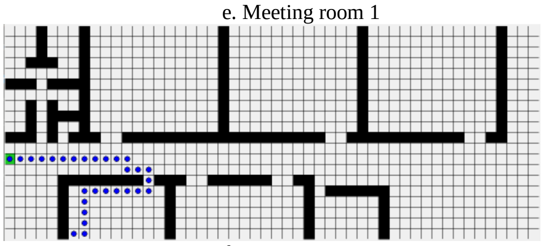

a. Room 1A

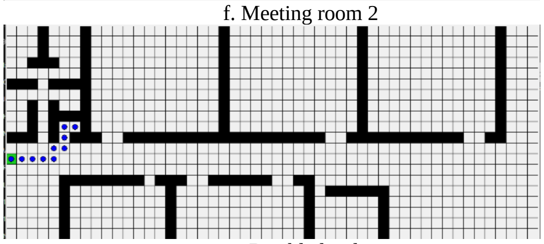

b. Room 1B

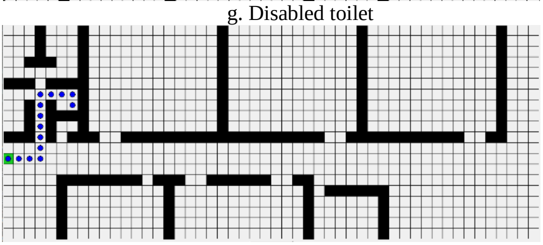

c. Room 1C

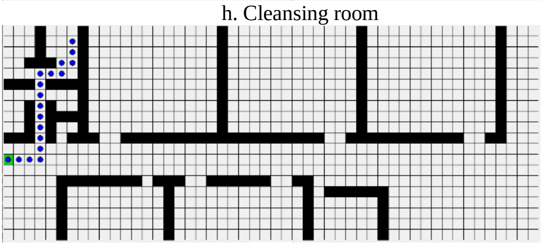

d. Power room

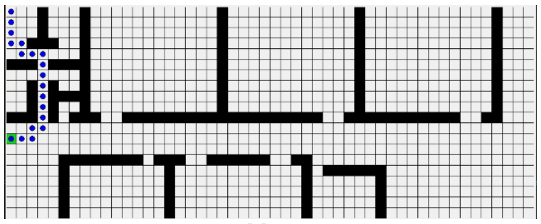

e. Meeting room 1


f. Meeting room 2


g. Disabled toilet


h. Cleansing room


i. Women restroom


j. Men restroom

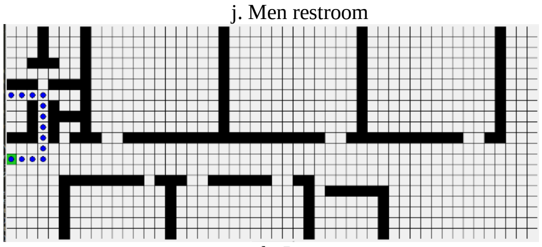

k. Pantry

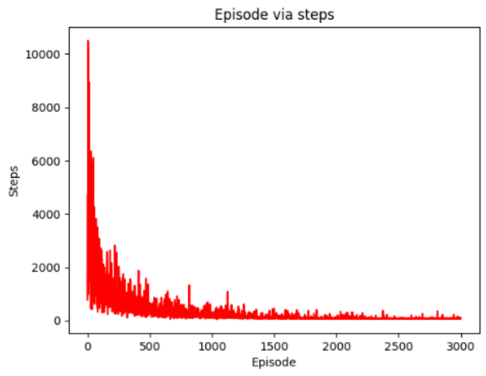
Figure 5. Shortest route Q-learning agent

The comparison of the path lengths generated by the Q-learning agent with the reference path lengths in each room can be seen in Table 6. The Q-learning agent succeeded in matching the length of the reference path or with a 100% accuracy rate. This demonstrates that the Q-learning agent has successfully built an optimal policy, allowing it to find the shortest path from the starting point to the destination.

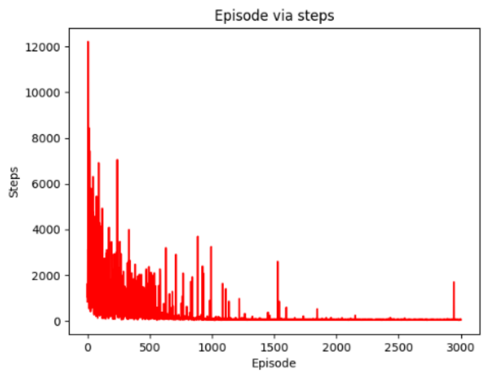
Table 6. Path length comparison

No.	Room Names	References Path (grid)	Q-learning Path (grid)	Q-learning Accuracy (%)
1	1A	63	63	100
2	1B	49	49	100
3	1C	20	20	100
4	Power room	41	41	100
5	Meeting room 1	34	34	100
6	Meeting room 2	27	27	100
7	Disabled toilet	9	9	100
8	Cleansing room	13	13	100
9	Women's restroom	18	18	100
10	Men's restroom	18	18	100
11	Pantry	12	12	100
	Average Q-learning Accuracy			100

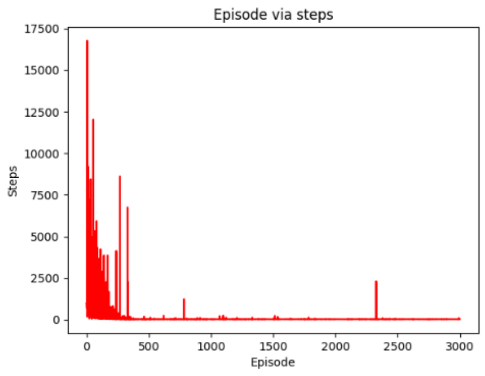
During the training process, the agent will accumulate knowledge from the DC building environment. The number of steps required by the agent to reach the goal gradually decreases as the number of episodes increases. Figure 6 a until k shows the number of steps taken by the agent to reach the goal from the starting point in the obstacle-free environment. The longer the distance from the agent's starting point to the destination, as in space 1A, the more episodes the agent needs to converge, while in room that are close to the goal, the fewer episodes it needs to converge.



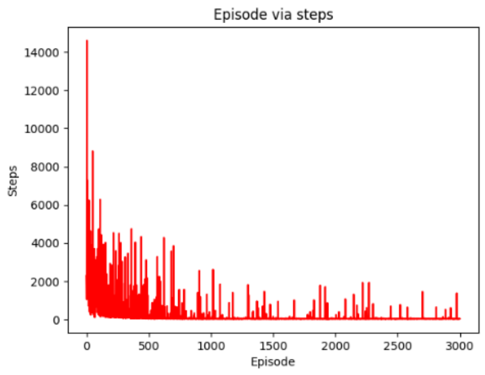
a. Room 1A



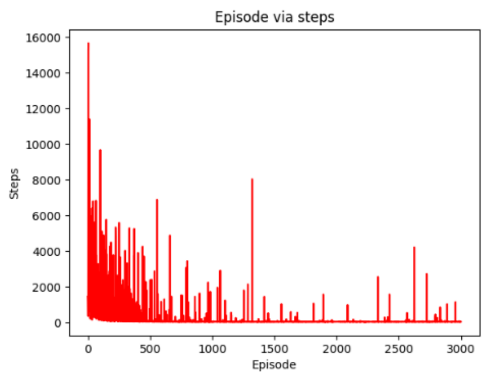
b. Room 1B



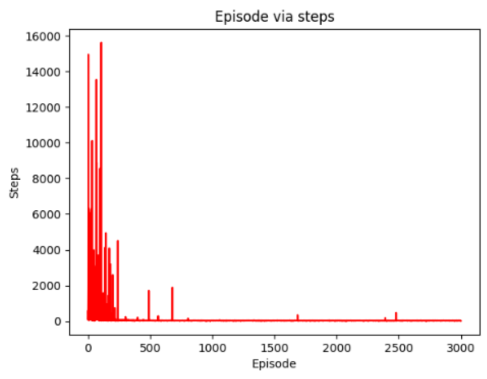
c. Room 1C



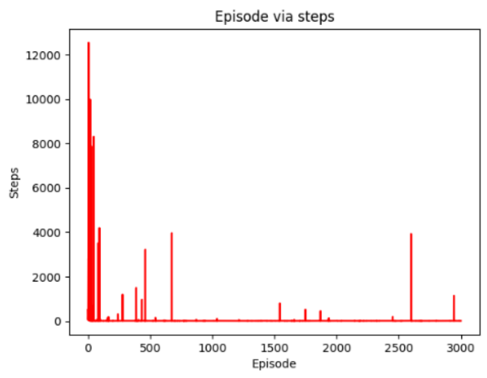
d. Power room



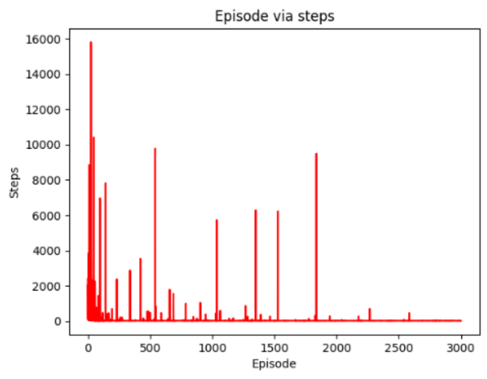
e. Meeting room 1



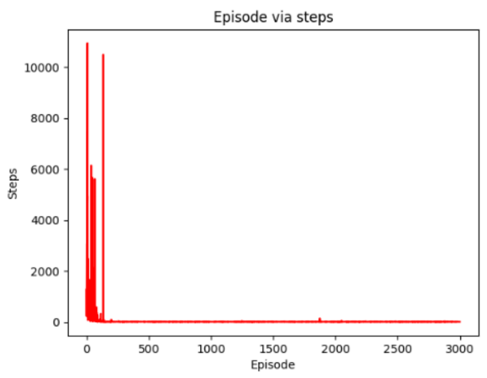
f. Meeting room 2



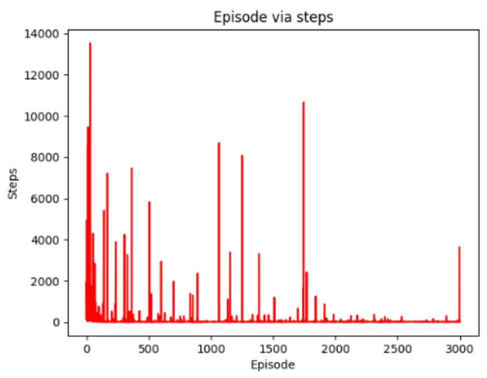
g. Disabled toilet



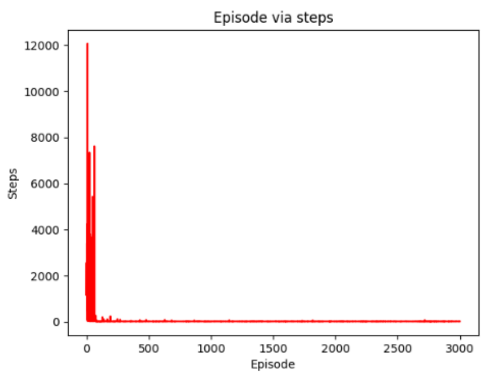
h. Cleansing room



i. Women’s restroom



j. Men’s restroom



k. Pantry

Figure 6. Q-learning convergence graph

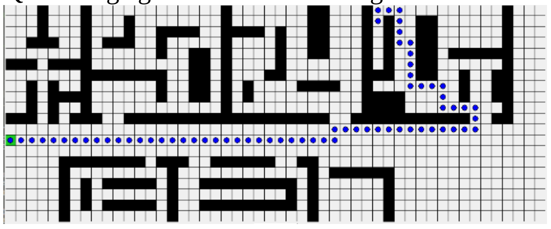
Table 7 shows at which episode the Q-learning agent begins to converge and the computation time required by the agent to complete 3000 episodes in the scenario of the obstacle-free environment. The average computational time required for the agent to complete one training iteration in an obstacle-free environment is 0.54 seconds.

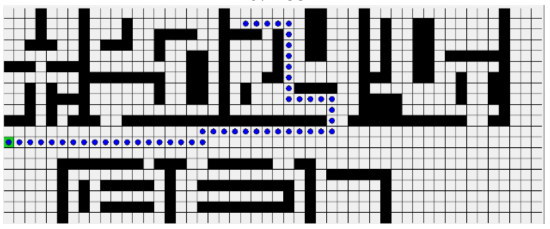
Table 7. Episode and CPU time Q-learning

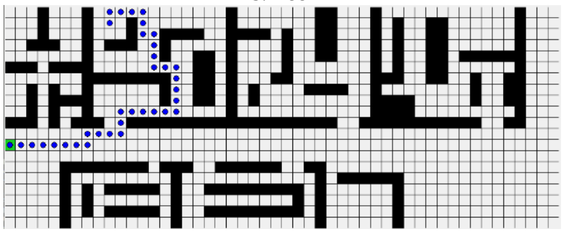
No.	**Room Names**	**Episode**	**CPU Time (s)**
1	1A	1042	1.53
2	1B	258	0.69
3	1C	152	0.22
4	Power room	641	1.28
5	Meeting room 1	238	0.84

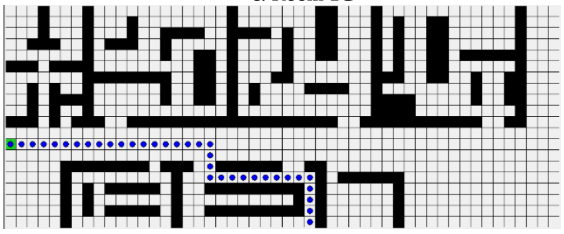
No.	Room Names	Episode	CPU Time (s)
6	Meeting room 2	231	0.53
7	Disabled toilet	24	0.14
8	Cleansing room	77	0.20
9	Women's restroom	153	0.19
10	Men's restroom	165	0.22
11	Pantry	83	0.16
	Average CPU Time		0.54

Testing was also conducted in an environment that has additional obstacles. The additional obstacles are placed in rooms 1A, 1B, 1C, and meeting rooms 1 and 2, representing furniture inside the rooms. The starting coordinate points used are the same as those in the obstacle-free environment. The resulting paths created by the Q-learning agent can be seen in Figure 7.


a. Room 1A


b. Room 1B


c. Room 1C


d. Meeting room 1

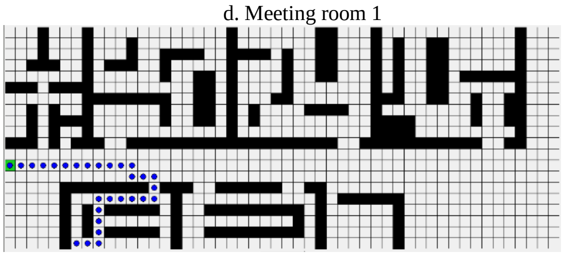

e. Meeting room 2

Figure 7. Q-learning paths in the environment with additional obstacles

Table 8 shows the path lengths generated by the agent to reach the goal. The agent can reach the goal even with additional obstacles and still obtain the shortest path.

Table 8. Path length of environment with additional obstacles and without additional obstacle

No.	Room Names	Q-learning Path length with additional obstacles	Q-learningPath length without additional obstacles
1	1A	65	63
2	1B	49	49
3	1C	34	20
4	Meeting room 1	34	34
5	Meeting room 2	27	27

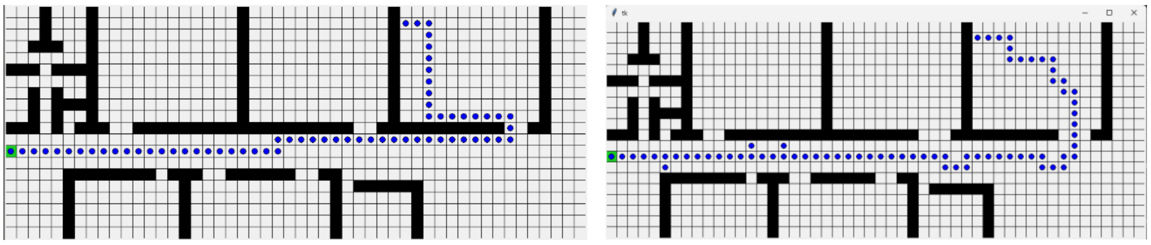
Table 9 shows the episodes required by the agent to begin converging and the computation time required to complete 3000 episodes in environment with additional obstacles. Q-learning successfully reached the goal in less than 3000 episodes with an average computation time of 0.76 seconds.

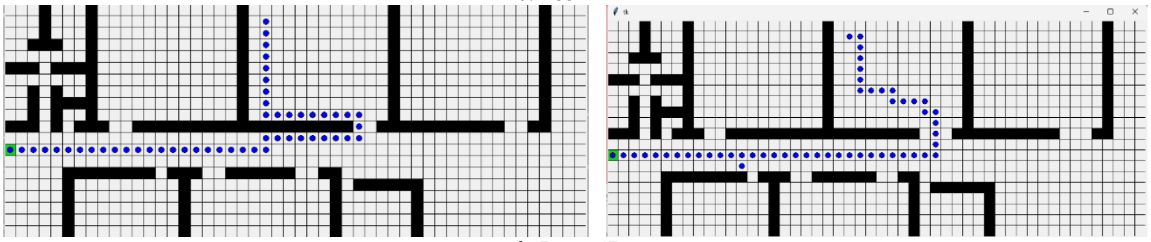
Table 9. Episode and CPU time Q-learning in an environment with additional obstacles

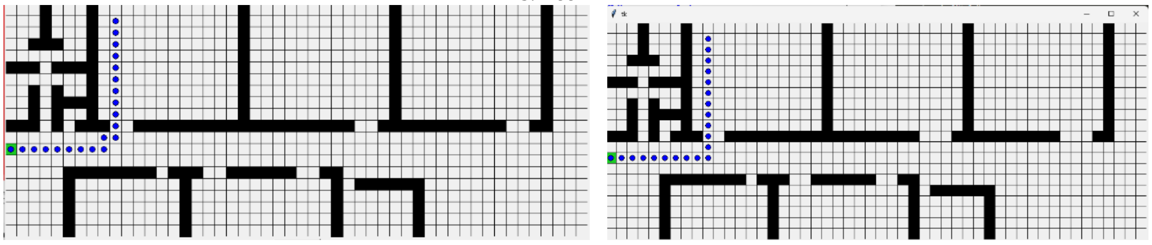
No.	Room Names	Episode	CPU Time (s)
1	1A	742	1.14
2	1B	342	0.95
3	1C	130	0.53
4	Meeting room 1	260	0.81
5	Meeting room 2	440	0.39
	Average CPU Time		0.76

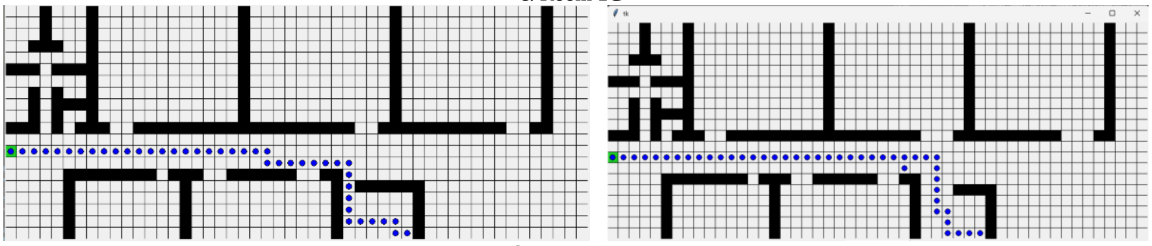
4.2. Comparison Q-learning with SARSA

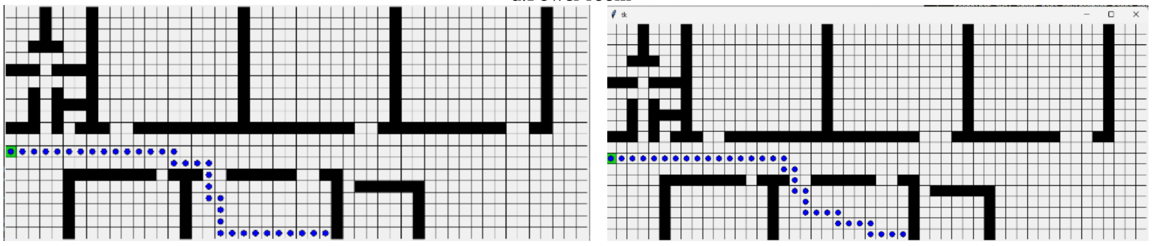
The Q-learning method used is also compared with SARSA using the same parameters and environment. A comparison of the paths created by Q-learning (left side) and SARSA (right side) can be seen in Figure 8 a to k.. There is a difference in the grid environment in the SARSA testing, specifically in room 1A, 1B, and the power room, where the room doors are made larger with a size of 3 grids. Meanwhile, in the Q-learning environment, the doors in room 1A and 1B are 2 grids in size, and the power room is 1 grid in size. This is because the SARSA agent cannot find a path if the room doors are made the same as the environment for Q-learning testing. Additionally, the SARSA agent maneuvers through obstacles to reach the exit door.

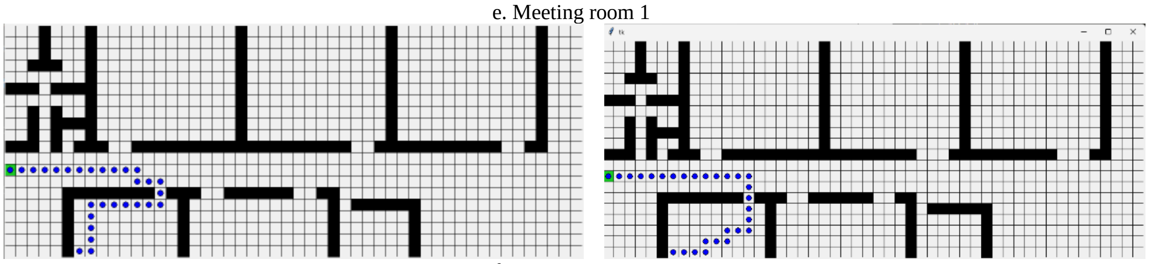

a. Room 1A

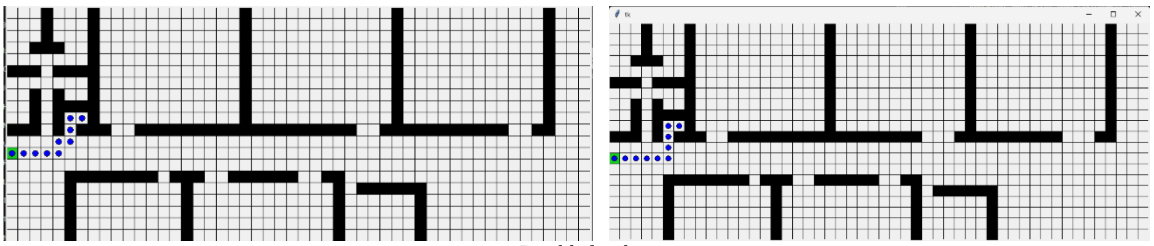

b. Room 1B

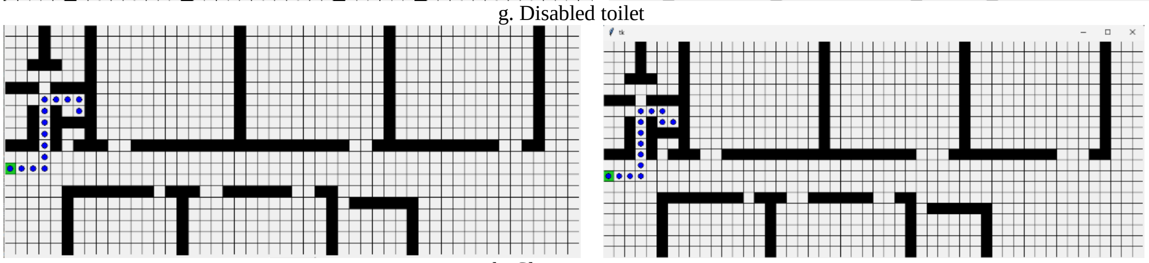

c. Room 1C

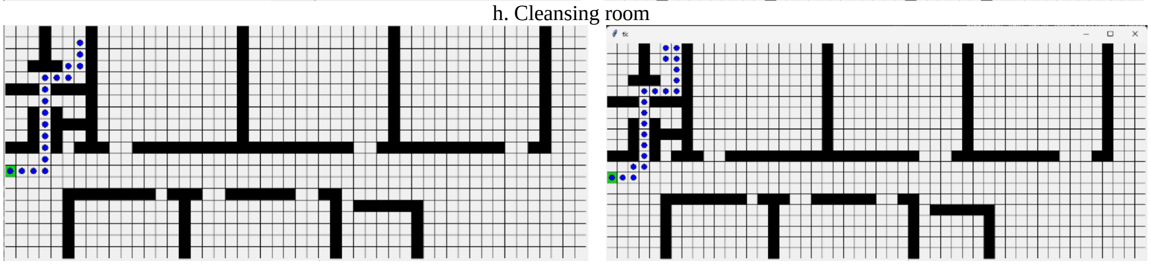

d.Power room

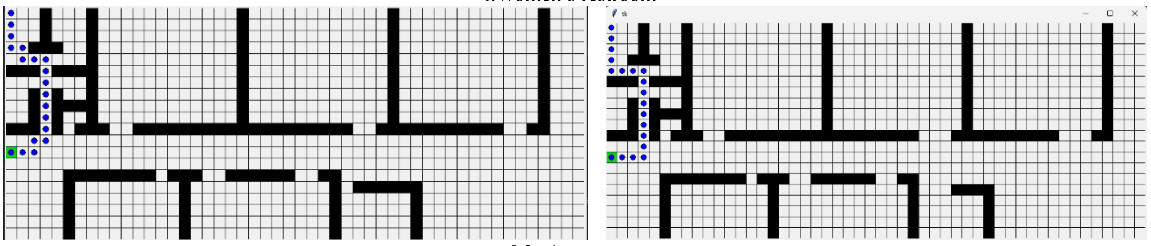

e. Meeting room 1


f. Meeting room 2


g. Disabled toilet


h. Cleansing room


i.Women's restroom


j. Men's restroom

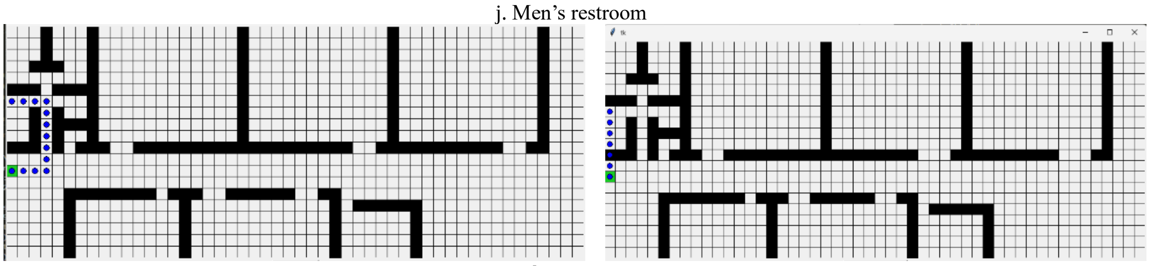

k. Pantry

Figure 8. Comparison of paths created by Q-learning (left) and SARSA (right) agents in each room

Table 10 provides a comparison of the path lengths created by the SARSA agent with those of the Q-learning agent. The path lengths generated by SARSA are generally longer than those of Q-learning, except for the disabled toilet where the path lengths are the same. Additionally, the path from the pantry is shorter for Q-learning, but the SARSA agent maneuvers through obstacles to achieve the shortest path, as seen in Figure 8 k. The average accuracy of the paths generated by the SARSA agent, when compared to the paths created by the Q-learning agent, is 75.05%. The longer path lengths produced by SARSA are attributed to the difference in approach in selecting policies. SARSA consistently adopts the same policy throughout the training process, even if there is a more advantageous policy available. Consequently, if the chosen policy is suboptimal, it can lead to longer path lengths. On the other hand, Q-learning always selects the policy with the maximum value for the next action, resulting in a more optimal policy.

Table 10. Comparison of SARSA and Q-learning path lengths

No.	Room Names	Q-learning Path Length (grid)	SARSA Path Length (grid)	SARSA Accuracy (%)
1	1A	63	77	77.77
2	1B	49	55	87.75

No.	**Room Names**	**Q-learning Path Length (grid)**	**SARSA Path Length (grid)**	**SARSA Accuracy (%)**
3	1C	20	30	50
4	Power room	41	45	90.24
5	Meeting room 1	34	60	23.53
6	Meeting room 2	27	31	85.19
7	Disabled toilet	9	9	100
8	Cleansing room	13	13	100
9	Women's restroom	18	22	77.77
10	Men's restroom	18	21	83.33
11	Pantry	12	6	50
	Average SARSA Accuracy			75.05

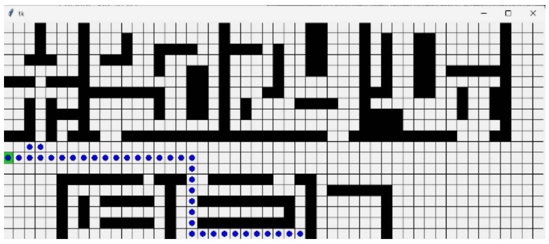
Table 11 presents a comparison of the episodes required to initiate convergence and the computational time needed to complete the training for both SARSA and Q-learning. Both methods can converge in fewer than 3000 episodes. In some rooms, SARSA may require fewer episodes than Q-learning. However, on average, the computational time for Q-learning is superior to SARSA, with 0.54 seconds for Q-learning compared to 0.94 seconds for SARSA.

Table 11 also shows that several rooms trained using Q-learning have a higher number of episodes but with shorter CPU time compared to training using SARSA, as seen in rooms 1A, Power room, Women's restroom, and Men's restroom. This indicates that Q-learning generally has a more efficient computation time per episode because it requires less computational time per episode than SARSA. This makes Q-learning advantageous in cases where many episodes are required, such as in environments with many obstacles, as many episodes can be completed in a shorter time.

Table 11. Comparison SARSA and Q-learning episode and CPU time

No.	**Room Names**	**Q-learning**		**SARSA**	
		Episode	**CPU Time (s)**	**Episode**	**CPU Time (s)**
1	1A	1042	1.53	554	1.56
2	1B	258	0.69	397	1.70
3	1C	152	0.22	153	0.70
4	Power room	641	1.28	639	1.44
5	Meeting room 1	238	0.84	380	2.11
6	Meeting room 2	231	0.53	280	0.83
7	Disabled toilet	24	0.14	47	0.30
8	Cleansing room	77	0.20	77	0.36
9	Women's restroom	153	0.19	104	0.69
10	Men's restroom	165	0.22	72	0.38
11	Pantry	83	0.16	108	0.23
	Average CPU Time		0.54		0.94

SARSA was also tested in an environment with additional obstacles. However, SARSA did not perform well in that environment. The SARSA agent could only create paths in meeting rooms 1 and 2, while in rooms 1A, 1B, and 1C, SARSA failed to find a path. The illustration of the paths created by the SARSA agent can be seen in Figures 9 a and b. The comparison of path lengths created by SARSA and Q-learning in an environment with additional obstacles can be observed in Table 11.


a. Meeting room 1

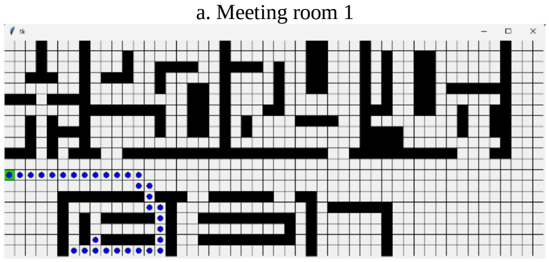

b. Meeting room 2

Figure 9. SARSA paths in the environment with additional obstacles

Table 12. Length of Q-learning and SARSA paths in environments with additional obstacles

No.	**Room Names**	**Q-learning**	**SARSA**
1	1A	65	-
2	1B	49	-
3	1C	34	-
4	Meeting room 1	34	44
5	Meeting room 2	27	87

The comparison of episodes required to initiate convergence and computational time to complete one training iteration in an environment with additional obstacles can be observed in Table 13. In Meeting Room 1, SARSA requires more episodes to start converging, specifically 487 episodes, and also has a longer computational time of 10.17 seconds. Meanwhile, in Meeting Room 2, SARSA requires fewer episodes, specifically 158 episodes, but has a longer computation time of 1.17 seconds.

Table 13. Comparison of episodes and computational time for Q-learning and SARSA in environment with additional obstacles.

No.	**Room Names**	**Q-learning**		**SARSA**	
		Episode	**CPU Time (s)**	**Episode**	**CPU Time (s)**
1	1A	742	1.14	-	-
2	1B	342	0.95	-	-
3	1C	130	0.53	-	-
4	Meeting room 1	260	0.81	487	10.17
5	Meeting room 2	440	0.39	158	1.17
Average CPU Time			0.76		

5. Conclusion

The Q-learning method has been successfully applied to design evacuation routes in the DC UNNES building on the 1st floor. The Q-learning method was tested in an environment without additional obstacles, comprising 11 rooms. The path created by the Q-learning agent can be observed in Figure 5. In terms of performance, the Q-learning agent successfully generated the shortest path, with a length equal to the reference path or having 100% accuracy, with the number of episodes below 3000 and an average computation time of 0.54 seconds. In this environment, as the distance from the agent's starting point to the goal increases, it requires more episodes and longer computation time. Then, in an environment with additional obstacles, Q-learning was tested in 5 rooms and still managed to find the shortest path with episodes below 3000 and an average computation time of 0.76 seconds.

The Q-learning method was also compared with SARSA in the same environment. The path length created by SARSA in an environment without additional obstacles has an accuracy rate of 75.05% and requires longer computation time, namely 0.94 seconds compared to Q-learning. However, the required episodes can still be below 3000. The paths created by SARSA in the pantry are indeed shorter than those created by Q-learning, but they are made by maneuvering through obstacles. In an environment with additional obstacles, SARSA performs less effectively. Out of 5 rooms, SARSA can only find paths in 2 rooms, namely meeting rooms 1 and 2.

Based on the experiments conducted, Q-learning works very well in the DC building environment. Q-learning can be an alternative method for evacuation route planning due to its excellent flexibility, applicable to various environments. Moreover, if there are changes in the environment, the Q-learning method can adapt quickly. Moving forward, we will continue to develop methods for evacuation route planning, especially using reinforcement learning, to make it more effective and efficient, such as using multi-agents, Deep Q-Networks, and developing methods for optimizing parameters like dynamic discount factors, reward systems, epsilon, and learning rates.

References

[1] K. Deng, Q. Zhang, H. Zhang, P. Xiao, dan J. Chen, "Optimal Emergency Evacuation Route Planning Model Based on Fire Prediction Data," *Mathematics*, vol. 10, no. 17, hal. 1–23, 2022, doi: 10.3390/math10173146.

[2] A. Karasi dan A. P. S. Rathod, "Finding safe path and locations in disaster affected area using Swarm Intelligence," *2016 Int. Conf. Emerg. Trends Commun. Technol. ETCT 2016*, 2017, doi: 10.1109/ETCT.2016.7882983.

[3] Y. Peng, S. W. Li, dan Z. Z. Hu, "A self-learning dynamic path planning method for evacuation in large public buildings based on neural networks," *Neurocomputing*, vol. 365, hal. 71–85, 2019, doi: 10.1016/j.neucom.2019.06.099.

[4] S. Xu *dkk.*, "Indoor Emergency Path Planning Based on the Q-Learning Optimization Algorithm," *ISPRS Int. J. Geo-Information*, vol. 11, no. 1, 2022, doi: 10.3390/ijgi11010066.

[5] Peraturan Menteri Kesehatan Republik Indonesia, "Permenkes RI Nomor 48 Tahun 2016 Tentang Standar Keselamatan dan Kesehatan Kerja Perkantoran." Kementrian Kesehatan, Jakarta, Indonesia, 2016. [Daring]. Tersedia pada: http://www.ncbi.nlm.nih.gov/pubmed/26849997%0Ahttp://doi.wiley.com/10.1111/jne.12374

[6] Y. Xue, R. Wu, J. Liu, dan X. Tang, "Crowd Evacuation Guidance Based on Combined Action Reinforcement Learning," *Algorithms*, vol. 14, no. 1, 2021, doi: 10.3390/a14010026.

[7] P. Thombre, "Multi-objective path finding using reinforcement learning," *Master's Thesis*, hal. 52, 2018, doi: https://doi.org/10.31979/etd.2ntb-4j8q.

[8] Y. Wu, J. Kang, dan C. Wang, "A crowd route choice evacuation model in large indoor building spaces," *Front. Archit. Res.*, vol. 7, no. 2, hal. 135–150, 2018, doi: 10.1016/j.foar.2018.03.003.

[9] P. Wu, Y. Wang, J. Jiang, J. Wang, dan R. Zhou, "Evacuation Optimization of a

Typical Multi-exit Subway Station: Overall partition and local railing," *Simul. Model. Pract. Theory*, vol. 115, no. October 2021, hal. 102425, 2022, doi: 10.1016/j.simpat.2021.102425.

[10] E. S. Low, P. Ong, dan K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Rob. Auton. Syst.*, vol. 115, hal. 143–161, 2019, doi: 10.1016/j.robot.2019.02.013.

[11] T. Li dan Y. Li, "A Novel Path Planning Algorithm Based on Q-learning and Adaptive Exploration Strategy," *2019 Sci. Conf. Network, Power Syst. Comput. (NPSC 2019)*, vol. 3, no. Npsc, hal. 105–108, 2019, doi: 10.33969/eecs.v3.024.

[12] A. Maoudj dan A. Hentout, "Optimal path planning approach based on Q-learning algorithm for mobile robots," *Appl. Soft Comput. J.*, vol. 97, hal. 106796, 2020, doi: 10.1016/j.asoc.2020.106796.

[13] L. Zhang, L. Tang, S. Zhang, Z. Wang, X. Shen, dan Z. Zhang, "A self-adaptive reinforcement-exploration q-learning algorithm," *Symmetry (Basel).*, vol. 13, no. 6, hal. 1–16, 2021, doi: 10.3390/sym13061057.

[14] A. Ardiansyah dan E. Rainarli, "Implementasi Q-Learning dan Backpropagation pada Agen yang Memainkan Permainan Flappy Bird," *J. Nas. Tek. Elektro dan Teknol. Inf.*, vol. 6, no. 1, hal. 1–7, 2017, doi: 10.22146/jnteti.v6i1.287.

[15] H. Park, D. Liu, dan S. Namilae, "Multi-Agent Reinforcement Learning-Based Pedestrian Dynamics Models for Emergency Evacuation Final Report," no. June, 2022.

[16] C. J. C. H. Watkins, "Learning from delayed rewards. PhD thesis," King's College London, 1989.

[17] R. S. Sutton dan A. G. Barto, *Reinforcement learning: An introduction, 2nd ed.* Cambridge, MA, US: The MIT Press, 2018.

[18] E. Even-Dar dan Y. Mansour, "Learning Rates for Q-Learning BT - Computational Learning Theory," 2001, hal. 589–604.

[19] N. Rochmawati, H. B. Hidayati, Y. Yamasari, H. P. A. Tjahyaningtijas, W. Yustanti, dan A. Prihanto, "Analisa Learning Rate dan Batch Size pada Klasifikasi Covid Menggunakan Deep Learning dengan Optimizer Adam," *J. Inf. Eng. Educ. Technol.*, vol. 5, no. 2, hal. 44–48, 2021, doi: 10.26740/jieet.v5n2.p44-48.

[20] M. S. Kim, J. S. Kim, M. S. Choi, dan J. H. Park, "Adaptive Discount Factor for Deep Reinforcement Learning in Continuing Tasks with Uncertainty," *Sensors*, vol. 22, no. 19, hal. 1–22, 2022, doi: 10.3390/s22197266.