Multi-agent Surveillance using A* Path-Search Algorithm and Q-Learning

Cem Ayder, Tiphanie Bent, Collin Makuza, Zofia Milczarek, Zaker Omargeel, Alisa Todorova, Vikram Venkat
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—This paper explores A* path-search algorithm and Q-learning, implemented in a multi-agent surveillance game. It focuses on how the effects of the variations of critical parameters and agent sets impact general learning performance. The research questions aim to determine the effects of the implemented algorithms on maps of different complexities. The results corroborated the hypothesis that was made for each of the parameters of the algorithm.**

***Index Terms*—Multi-agent Surveillance, Q-Learning, Ms. Pacman, Reinforcement Learning, A*, A-star, Path-Search Algorithm, Q-table, Q-values**

I. Introduction

Interest in mobile automatic surveillance, together with path planning technology, has been rapidly-growing in the research field of both Artificial Intelligence and Robotics. Here, the term *mobile agents* refers to robots, such as surface robots, and flying robots. This is due to the broad spectrum of potential applications in security tracking, monitoring, and autonomous navigation, especially for military and industrial surveillance systems. Path planning technology can help the agent to successfully and safely (i.e. without colliding with obstacles) finish its assigned tasks within the allocated time [6]. Some state-of-the-art Multi-Agent Path Finding (MAPF) algorithms are A*-based algorithms, reduction-based algorithms, and dedicated search-based algorithms [2]. This paper explores A* path-search algorithm and Q-learning.

The research questions this paper aims to answer are the following:

- How do variations of the Q-learning parameters impact the agent's learning?
- Is there a discernible effect of the markers on the intruder win ratio?

This paper is organized as follows: First, section II-A explains the Multi-agent Surveillance Game, the two teams of agents, and the two markers - Trace and Yell. Next, section II-B describes in detail the implemented algorithms developed for this project. The data sets used for the experiments are explained in section II-C. In section III our implementation of the said methods is explained, as well as the creation of the Graphical User Interface. A detailed description of the experiments is given in section IV. In V all results from the experiments are presented, which are then interpreted and explained in the section VI. Finally, in section VII a summary of this paper is provided.

II. Method

A. Multi-agent Surveillance Game

There are two types of multi-agent surveillance agents, discussed in this paper: intelligent surveillance agent (SAs, also known as guards) and intelligent intruder agent. The guards have two main goals: explore the map and capture the intruders. However, the intruders have to only find a given target location before getting caught by the guards.

Both the guards and the intruders don't hold knowledge about the map upon spawning. That is why they explore their environment and learn it with the help of A* path-search algorithm and Q-Learning, as explained in section II-B1 and section II-B2, respectively.

The guards can help each other with the exploration of the map, but only by indirect communication. Indirect communication is also known as stigmergy-based (stigmergic) or pheromone-based communication. The team of guards have 2 different types of markers: Trace and Yell.

Trace changes colors based on what the agent perceives. The following tree patterns are created:

- If the agent perceives *no opponents*, then the trace is green.
- If the agent perceives *only one opponent*, then the trace is orange.
- If the agent perceives *two or more opponents*, then the trace is red.

The level of threat is later connected to the reward table, thus, ensuring a multi-agent insights on the learning. The agents' subjective perception of the world takes into account the signals left by their teammates. Trace possess a *lifeTime* parameter that we later experiment on (see section IV).

Yell is an instantaneous information, which is only available for a certain *radius* during a single time step.

Both markers allow the transformation of the information that was sensed, into structured knowledge, contributing to the decision making of each agent.

B. Algorithms

1) A path-search algorithm:* A* is an informed search algorithm that finds the shortest path between a specified starting point and a specified goal. The shortest path is determined by the path cost and by the heuristic function. The most common heuristic function used for A* and especially for a square grid is the Manhattan Distance. It is calculated as the sum of the absolute differences between the points. For example, Manhattan Distance of the two points (x1, y1) and (x2, y2) is the following:

$$|x1-x2| + |y1-y2| \tag{1}$$

It gives an estimate of the minimum cost from any point n to the goal. See the pseudocode in algorithm 1.

Algorithm 1 A* Search Algorithm

```
 1: Initial open list (OL) with only starting node
 2: Initial an empty closed list (CL)
 3: while the goal has not been reached do
 4:     C = the node with lowest f-score in OL
 5:     if C is our destination then
 6:         we are done
 7:     else
 8:         put C in CL and loop all neighbours (N)
 9:         if N has lower g-value than C and is in CL then
10:             replace N with the new g-value
11:             make C N's parent
12:         else g-value of C is lower and N is in OL
13:             replace N with the new g-value
14:             make C N's parent
15:         end if
16:         if  N is not on both CL and OL then
17:             add N to OL and set g-value
18:         end if
19:     end if
20: end while
```

2) Q-learning: The Machine Learning approach used in this paper is Q-learning. It is a Reinforcement Learning algorithm, used to derive the best possible policy in a given environment, i.e. the sequence of actions that will ensure the highest gain in rewards. Q-Learning compares the expected utilities of the available actions to find the action which has the highest q-value at a given state:

$$U(s) = \max_a Q(s, a) \tag{2}$$

where $Q(s, a)$ is the Q-value of action a at state s.

The Q-learning agent aims to learn the expected reward (Q-value) from each state-action [3]. The acquired knowledge is stored as [states, actions] in a Q-table, which is constantly updated as the agent learns. In this paper the states and actions used for Q-Learning are as follows:

- **State(s)**: The agent's starting position (i.e. initial tile).
- **Actions(s)**: The agent moves to a new tile. Valid moves are up, right, down and left. Therefore, on an $m \times n$ map, our Q-table will comprise of $4 \times m \times n$ cells.

See the pseudocode in algorithm 2.

Q-values are first initialized to 0. During the learning phase, they are iteratively updated by the Bellman Equation:

$$Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha \times TD(s_t, a_t) \tag{3}$$

Algorithm 2 Q-Learning Algorithm

```
1: Initial $\alpha$, $\gamma$ and Q table
2: if episode $\leq$ max episode return Initial $s_t$ then
3:     while $s_t$ is not terminal do
4:         Choose $a_t$ using policy derived from Q table
5:         Execute action $a_t$, observe R($s_t$, $a_t$) and $s_{t+1}$
6:         Update Q(s,a) by using Bellman Equation
7:         $s_t$ = $s_{t+1}$
8:     end while
9: end if
```

where:

- s_t is the state at time t.
- a_t is the action at time t.
- α is the learning rate of the agent.
- *TD* is the temporal difference.

Temporal Difference (TD) is a method that provides the value with which the Q-value at the previous state should be updated after performing an action. The function is as follows:

$$TD(s_t, a_t) = r_t + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{4}$$

where:

- $Q(s_t, a)$ is the Q-value of action a at state s at time t.
- r is the reward given to agent for reaching a new state s.
- γ is the discount factor, i.e. how much the agent values future rewards over immediate awards.
- $\max_a Q(s_{t+1}, a)$ is the highest Q-value available at the state s at time $t + 1$.

C. Data sets

Map complexity is defined by the amount of teleports, walls, enclosed areas, and shaded areas. The more elements of this kind exist, the more difficult it is for the agent to gain knowledge. Thus, we can measure the impact of increasing map complexity over the agent's winning rate. For the experiments 10 different maps of varying levels were prepared. They can be grouped under were three general difficulties: easy, intermediate and hard. The intent behind designing maps with different levels of difficulty was to produce varied environments to perform experiments on.

The difficulty level depends on the number of objects present in the map. In general, increasing the amount of obstacles is expected to slow down the agent progression. Thus, the time steps required to acquire the knowledge is larger. An easy map is defined as having no shaded areas, at least 4 walls, at least 1 enclosed area and at least 2 teleports. An intermediate map is build from an easy one by adding more of one specific component. By doing so, we are able to perform a sensitivity analysis to detect which kind of obstacles impacts the winning rate the most. Yet, such intermediate map is overall not very dense, and the path to the goal is considered not too long. An advanced map contains an equal amount of each obstacle type, but has between two to four times the amount as the intermediate map.

III. Implementation

A. Path finding algorithm: A*

In this paper the A* Algorithm is used by agents to find the shortest path between two points in a specified map.

B. Ray Casting

Ray Casting is a popular graphic rendering technique from the 90s. In this paper it is used to render a semi-3D model of the simulation [7].

We have created an overhead view of the ray casting, which shows ray collisions, implemented along with generated obstacles from a map, through the file parser. The rays are generated from the eye of the observer (origin position of the agent) to sample the *light*, i.e. the pre-defined vision range. The general idea is to find the closest object blocking the path of the ray.

The *Calcrays* method casts the vision range of the agent by using the Line2D object as generators and storing the vision in a Linked List. The vision of the agent can also be obtained by getting the Visible tiles, which is used for implementing the markers. Additionally, a polygon is also created as a boundary around the casted rays, clearly defining use cases for encountering of obstacles. See the pseudocode in algorithm 3.

Algorithm 3 Polygon Boundaries

```
1: count ← 0
2: for each side in polygon do
3:     if rayIntersectsSegment(P, side) return count ← count + 1 then
4:     end if
5:     if isOdd(count) return inside
         else return outside then
6:     end if
7: end for
```

C. Q-learning

Q-Learning was chosen because the algorithm will converge to an optimal policy in an environment with finite states and actions such as is the environment in this project [4]. Additionally, this method can adapt to any map complexity given. Furthermore, with the use of Q-Learning, many aspects of our game were easier to implement as they were handled by the learning process. For example, we did not explicitly implement the agent to be unable to walk into walls, but rather, we gave it a severe punishment for that, and it quickly learned to avoid walls.

The algorithm starts by initializing every entry in the Q-table to 0. Subsequently, the agent enters into the learning phase during which it will play the game until it gets to a terminal state for a given number of cycles. During the learning phase, the agent is acquiring rewards for each performed action and updating the Q-value for the said action in its Q-table [5].

1) Reward Table: The values of the reward are stored in an $m \times n$ table, where m and n are the dimensions of the given map. Each cell of the table stores the amount of reward points given to the agent for attaining the corresponding position. The agent is rewarded for reaching a position that is closer to the goal. Therefore, the reward r given to an agent for attaining position P on our map is determined by:

$$r = -1 \times dist(P, G) \tag{5}$$

where G is the position of the goal and $dist(A, B)$ is a function that outputs the distance between points A and B. From this formula follows that as the distance decreases, the reward given to the agent increases.

Furthermore, we want to attribute a significantly higher reward to the agent for attaining the goal and a significantly lower reward for walking into a wall tile. Therefore, we assign values 1000 and -1000, respectively, at the appropriate positions in the reward table.

2) Epsilon-Greedy Algorithm: The Epsilon-Greedy Algorithm is a method implemented in Q-Learning to allow the agent to explore its environment. Indeed, for the agent to accurately approximate the Q-values, it will have to explore as many state-action pairs as possible. Thus, we set up a randomness level epsilon in [0,1] which describes the probability that the agent will perform a random move at each state.

D. Markers

The markers are implemented both as an instance of each agent and as information that can be stored on a Tile. Ultimately, they are connected to the general architecture via the Q-value table of the Q-learning.

In order to create the Yell marker, a general Audio Object class was created that contains the 3 vital variables, namely the radius, agent, and its position. The logic behind this marker lies in the iteration through the agent list. The respective distances between the agents are checked to the range of the yell radius, and if true, then the agent follows the yell. There is also a priority list implemented between the markers, with yell having more importance. Additionally, while the agent follows the yell and encounters another agent following the yell, the priority is switched to follow the latter. Hence, the priority decreases in the order of seen intruders, yell and trace.

E. Graphical User Interface (GUI)

The GUI implemented is in 3D for better visualizations through the use of the LWJGL (Lightweight Java Game Library) library, and the menus are implemented in JavaFX. After loading the map, it is possible to set the Q-learning parameters and start the simulation. Additionally, it is also possible to make changes to the text map. The instructions for the visuals are provided on the game screen.

IV. Experiments

The majority of the experiments were related to varying the parameters of the Q-learning algorithm and the marker variables, and the subsequent data sets were compared after being implemented on maps for different complexities. Additionally, the number of guards and intruders was also tweaked to test the winning rate between the 2 sets of teams. The experiments pertaining to the yell were varied over values consisting of the range between 20 to 50 with increments of 5, and were run on a map of greater complexity. The Q-learning section of the experiments consisted of testing win ratios over changes in the Discount Factor, Randomness Level and Learning rate. They were visualized through plots of learning cycles against the move counts required.

V. Results

The inferences from the Q-learning experiments and the graphs are outlined below. Regarding the Learning rate, when increasing it's value, there is a considerable decrease in the number of moves played, with less learning cycles. Additionally, as Discount Factor increases, the move count spikes, with mostly increasing increments, and the learning cycles increase for higher discount factor values as well. As the Randomness Value increases, both the moves and learning cycles required spike immediately. However, there seems to be an aberration for one tested value of 0.6, where both parameters are very low over successive learning cycles. Iterations of the yell radius variations showed that as the radius increased (on a map of average complexity), the intruder win ratio lowered. More information on these data sets can be found in section IX.

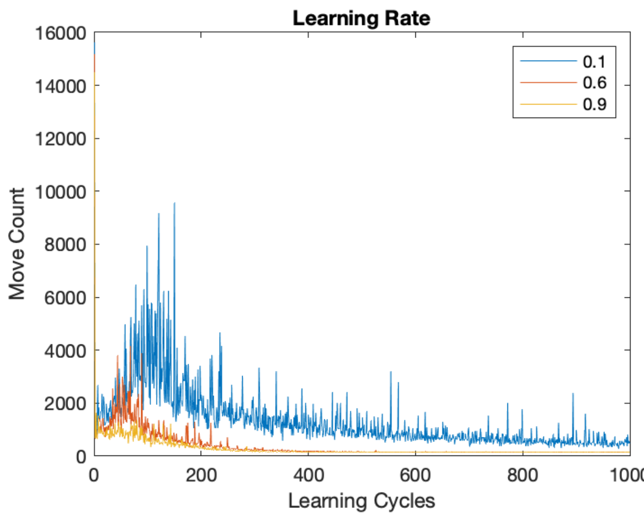

Fig. 1. Learning rate comparison between learning rates: 0.1, 0.6 and 0.9

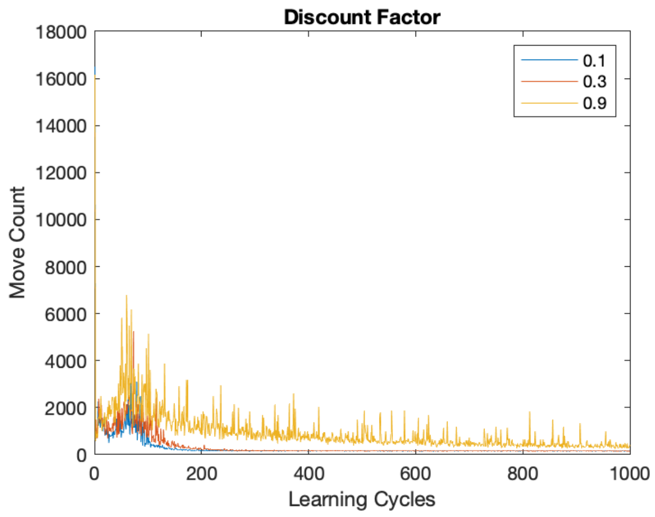

Fig. 2. Discount factor comparison between discount factors: 0.1, 0.3 and 0.9

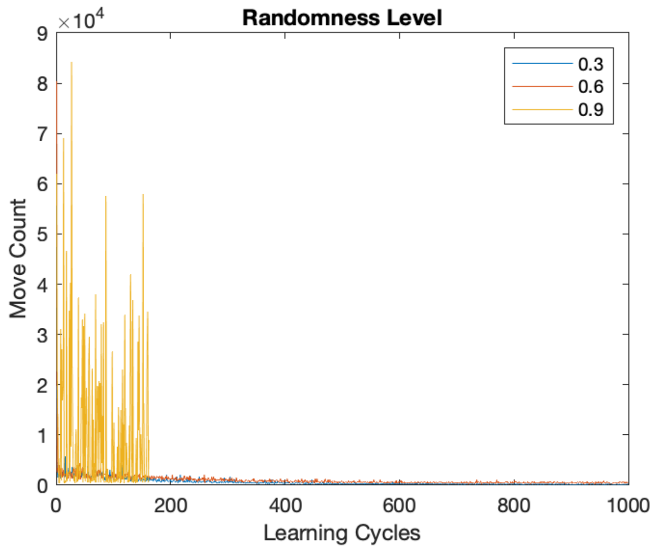

Fig. 3. Randomness level comparison between randomness levels: 0.3, 0.6 and 0.9

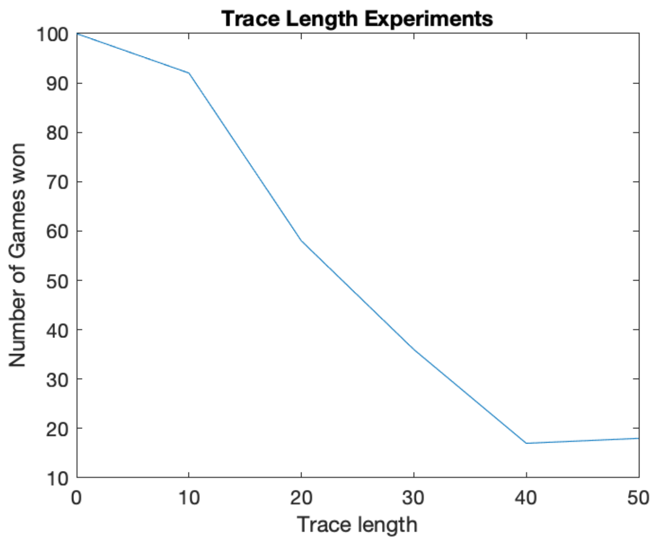

Fig. 4. Trace Length Experiments

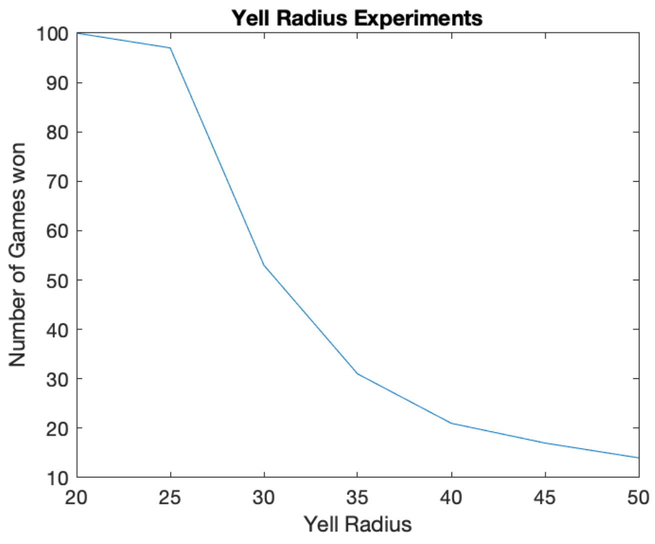

Fig. 5. Yell Radius Experiments

VI. Discussion

The results of the experiments on the effect of the Q-learning variables on the agents' learning demonstrates that the performance is positively correlated with the value of the learning rate and negatively correlated with the discount factor. As for the randomness level, the data suggests that given a value around the bounds of [0, 1], the agents performance is negatively affected, whilst given a value between [0.3, 0.6] will positively affect the agent.

The discount factor quantifies the importance the agent gives to future rewards relative to immediate rewards. In other words, as this value increases, the agent will be more inclined to take a path that first does not maximize its reward gain to eventually get to a higher reward. Therefore, as this value is increased, the agent will base the quality of every move, on further rewards, therefore the agent will opt for paths that require a larger number of moves to reach the goal.

It was expected that the agent's performance would be positively impacted by the increase in learning rate as this would translate into the agent building its policy with less learning cycles.

The results on the experiments on the randomness level were as expected. As this value approaches 0, the agent does not explore new paths and thus, does not discover better paths to the goal than the ones he discovers during the first learning cycles. In contrast, as ϵ approaches 1, the intruder makes increasingly random moves, and eventually the algorithm during the learning phase slows down as the intruder is in essence finding the goal by brute-force.

The results from the experiments performed on the yell and trace ranges also confirm our initial hypothesis that as they increase, the guards are able to catch the intruder, and the ratio of guard wins keep increasing with the linear increase of the markers.

All these results thus are instrumental in answering the two research questions.

VII. Conclusion

The paper aimed to address the following 2 research questions:

- How do variations of the Q-learning parameters impact the agent's learning?
- Is there a discernible effect of the markers on the intruder win ratio?

After performing experiments on the Q-learning parameters, an increase in the learning rate had a positive correlation with the agent's performance, and as the randomness value decreased, the agent stopped exploring new paths. The discount factor should be minimized, since the agent will opt for longer paths with higher values. After testing the yell and trace ranges, the guards were able to capture the intruder as their respective values increased, hence acting as an effective marker for the guards to win.

Some recommendations for further study and research would be implementation of multi-agent Q-learning algorithm when direct communication is allowed. In multi-agent environment the agents interact with the environment and even can collaborate, coordinate, or compete with each other, as well as collectively learn to accomplish a particular task [1].

VIII. References

References

[1] Lucian Busoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 38(2):156–172, 2008.

[2] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7627–7634, Jul. 2019.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[4] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[5] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[6] Meng Zhao, Hui Lu, Siyi Yang, and Fengjuan Guo. The experience-memory q-learning algorithm for robot path planning in unknown environment. *IEEE Access*, 8:47824–47844, 2020.

[7] Karel J Zuiderveld, Anton HJ Koning, and Max A Viergever. Acceleration of ray-casting using 3-d distance transforms. In *Visualization in Biomedical Computing'92*, volume 1808, pages 324–335. International Society for Optics and Photonics, 1992.

IX. Appendix A: Results

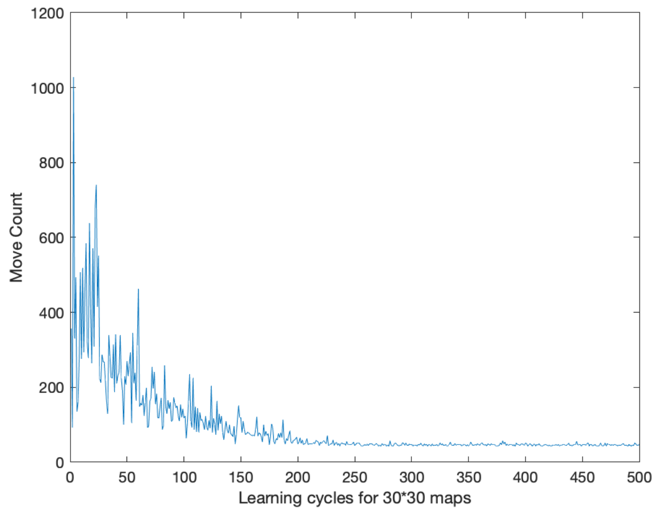

Fig. 6. Learning cycles for 30x30 maps

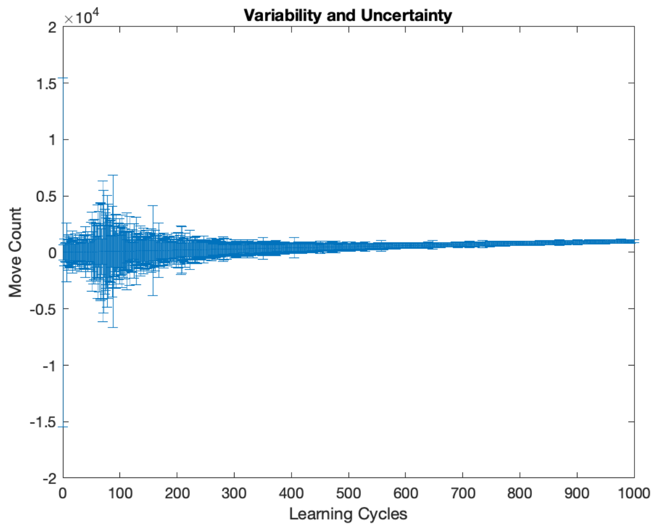

Fig. 8. Error bar interval of move counts

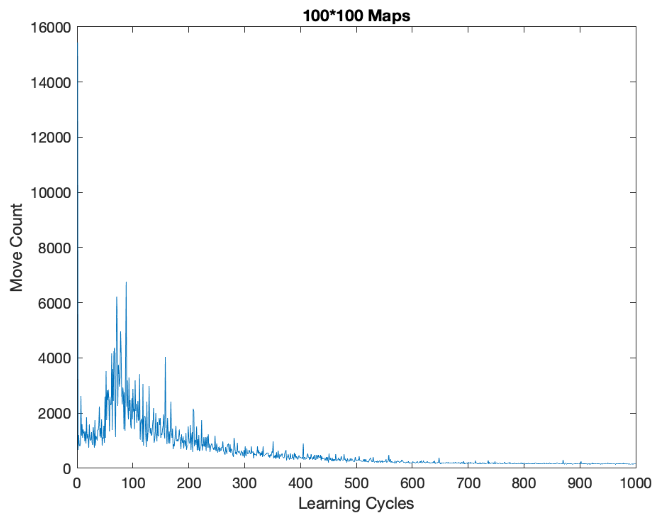

Fig. 7. Learning cycles for 100x100 maps

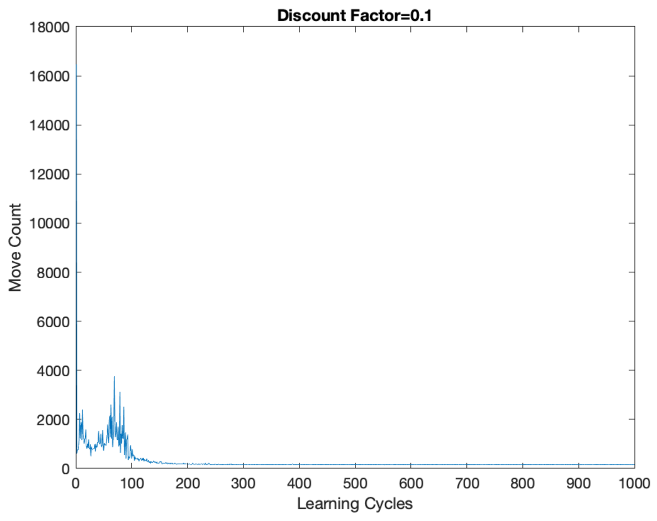

Fig. 9. 0.1 discount factor

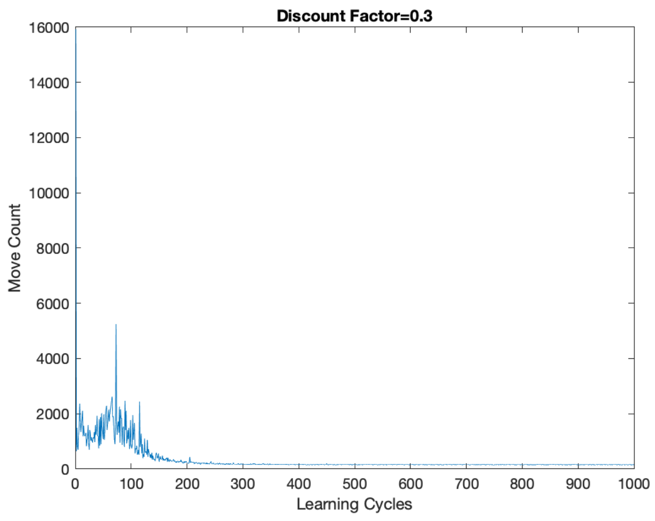

Fig. 10. 0.3 discount factor

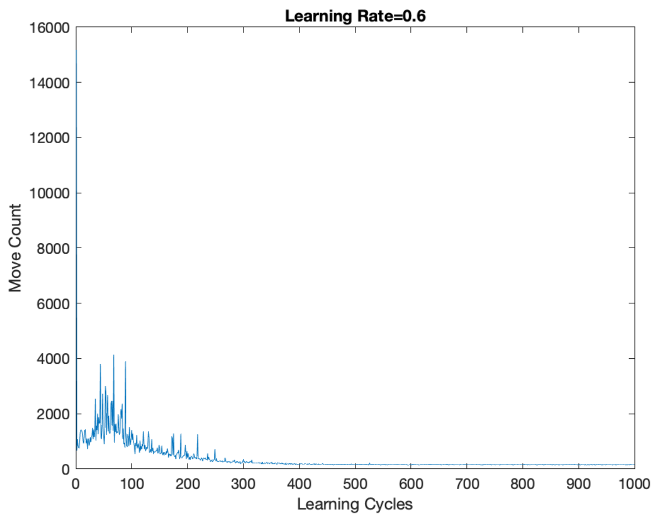

Fig. 13. 0.6 learning rate

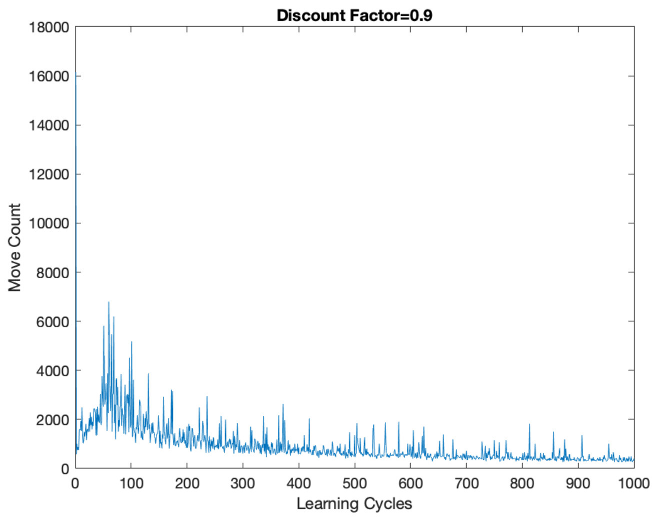

Fig. 11. 0.9 discount factor

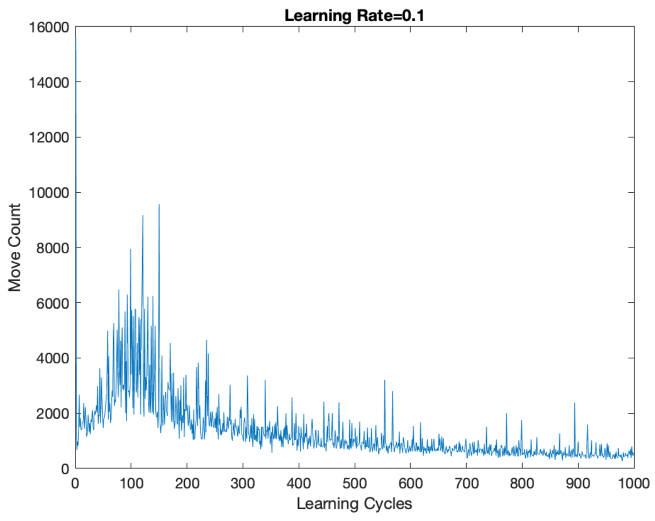

Fig. 12. 0.1 learning rate

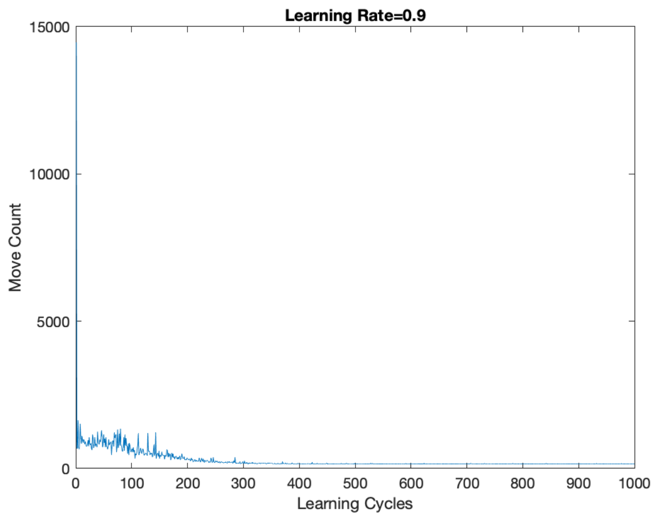

Fig. 14. 0.9 learning rate

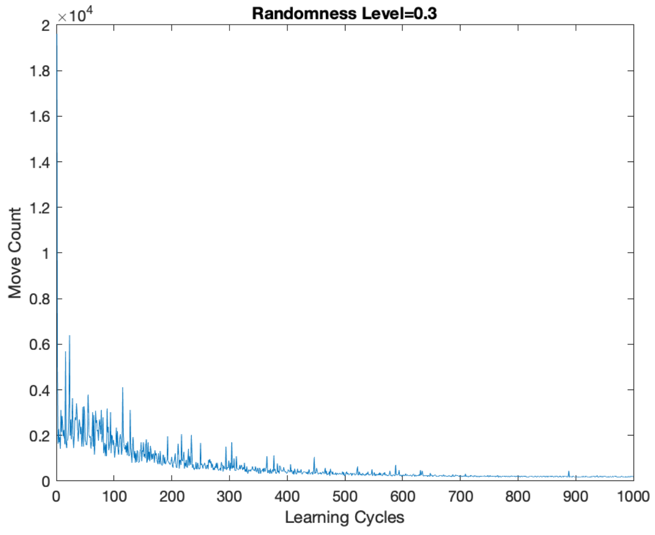

Fig. 15. 0.3 randomness level

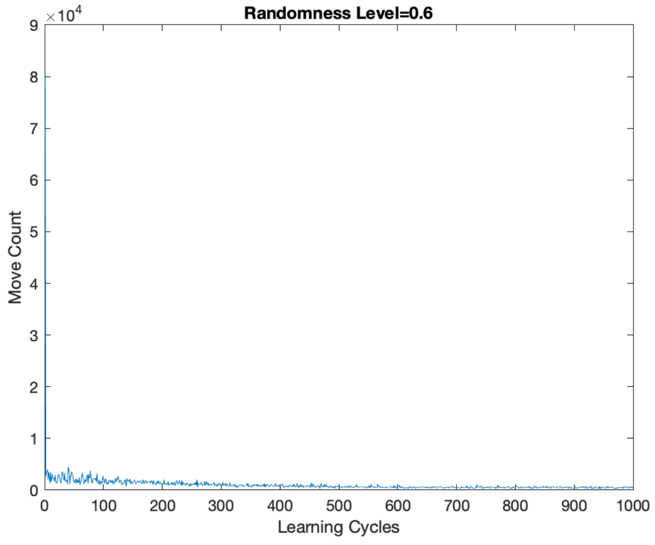

Fig. 16. 0.6 randomness level

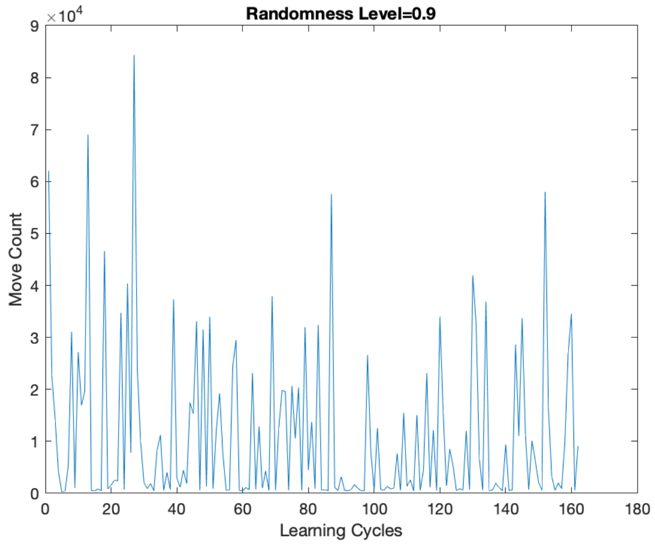

Fig. 17. 0.9 randomness level

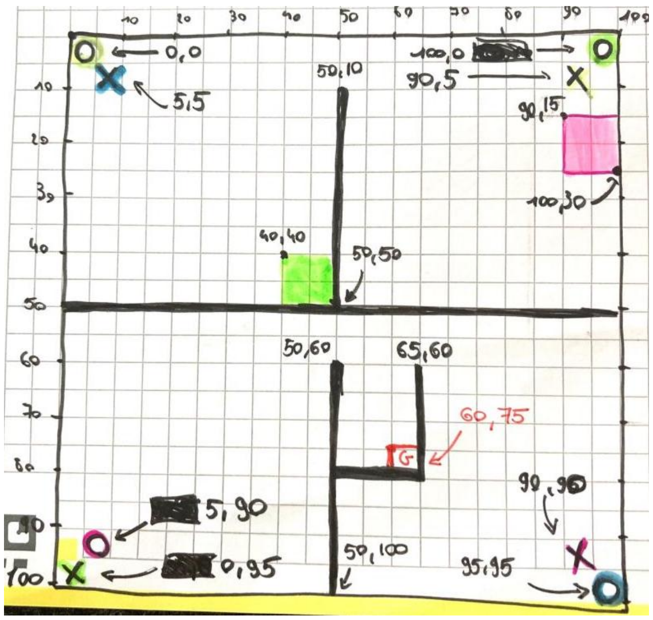

Fig. 18. Example of an easy map

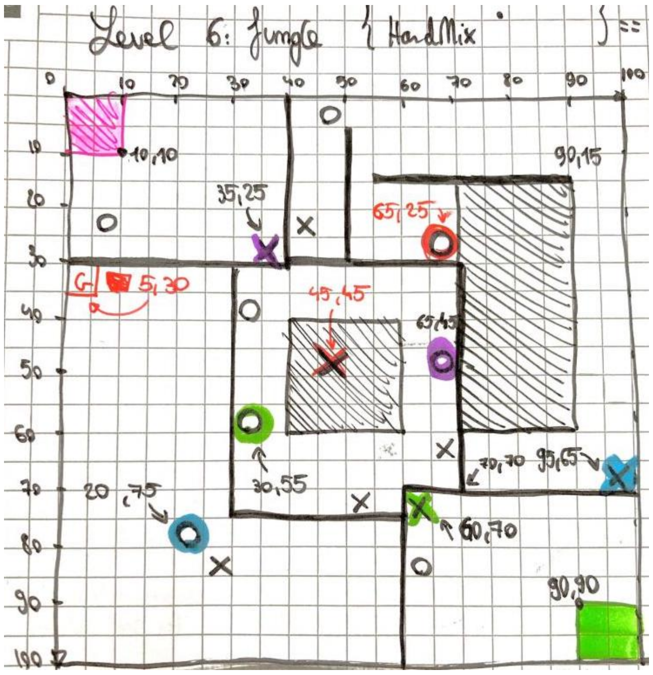

Fig. 19. Example of an intermediate map with enclosed areas

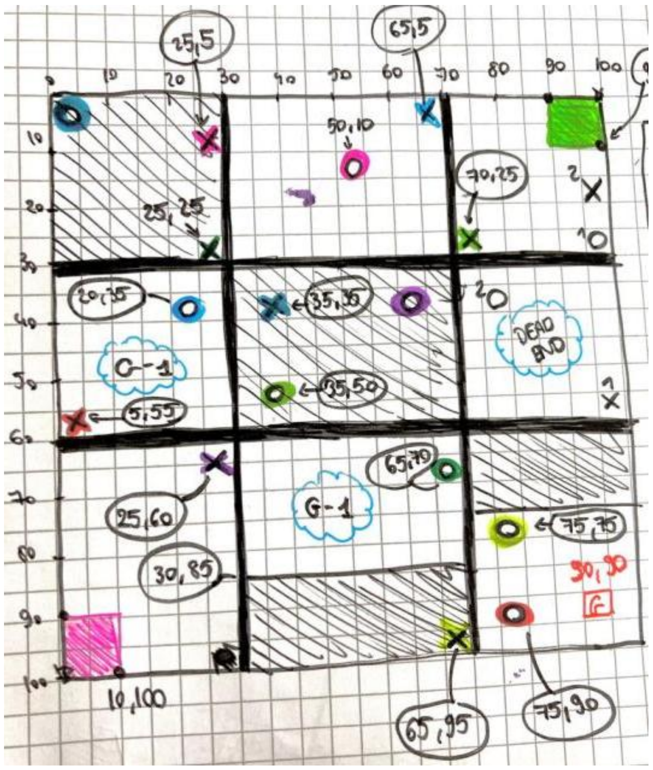

Fig. 21. Example of a hard map

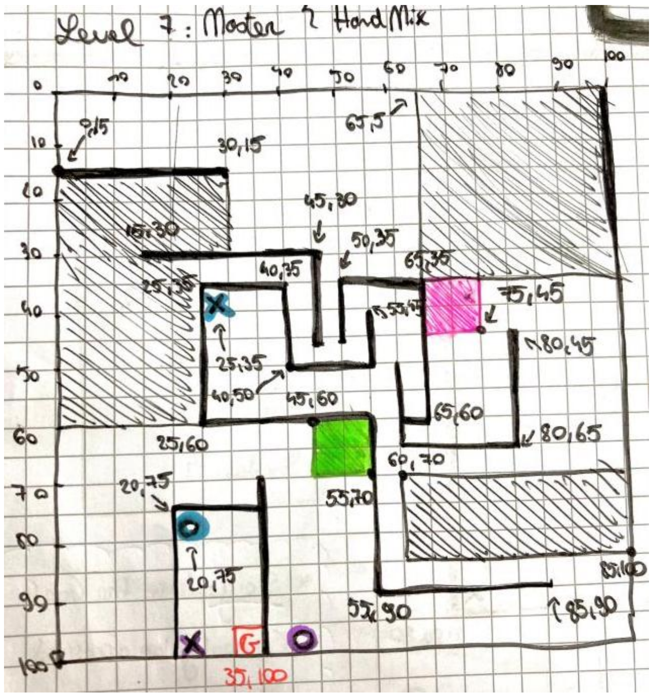

Fig. 20. Example of an intermediate map with many walls