Intelligent Robotic Systems Project Report: avoiding black areas with QLearning

Andrea Rettaroli

November 2022

Summary

1 Problem Introduction

The goal of the project is to design and implement a system composed by a robot and an arena where the robot avoids the black parts of the unknown environment as much as possible. Basically the robot must try to stay as much as possible in the white areas of the environment/arena, to do this we want to minimizing the passage time on the black areas. In a real scenario the black areas could indicate a danger or in any case areas that the robot must avoid as much as possible. Various types of scenario that rappresent this problem situation will be simulated and tested using ARGoS.

2 Problem Analysis

To achieve this goal and minimize the time of stay in the black areas of the arena by the robot we will approach the problem through reinforcement learning using the Qlearning algorithm.

2.1 QLearning

Reinforcement Learning is a process characterized by two entities: an environment which simplifies a certain reality and an agent that interacts with the environment. The following diagram describes the process.

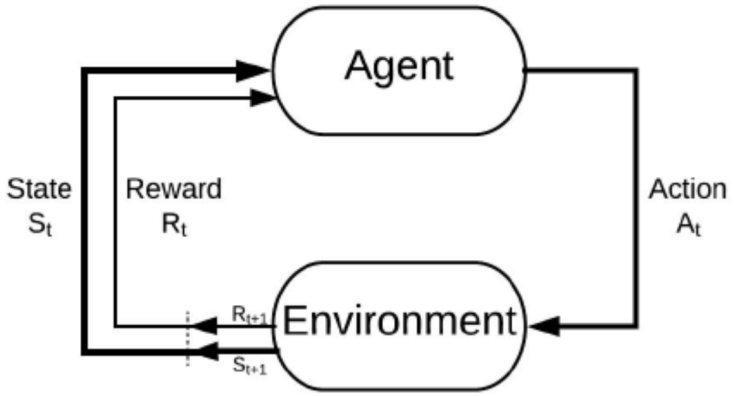

Figure 1: There are 4 basic components in Reinforcement Learning; agent, environment, reward and action.

In our use case the robot is the agent and the arena is the environment. The agent in this case not know the map, but it must avoid the black areas as much as possible and stay in the white ones. To do this as best as possible he can use a classic approach based on calculate the expected value of future

rewards for each action in each possible state of the environment; this allow the agent to choose the best action to do in each state. This technique based on reinforcement learning can be adapted to solve our problem. The core aspect of this technique is the Q-Table that is the data structure used to calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action to do at each state. The following diagram describes the Q-Table populating algorithm.

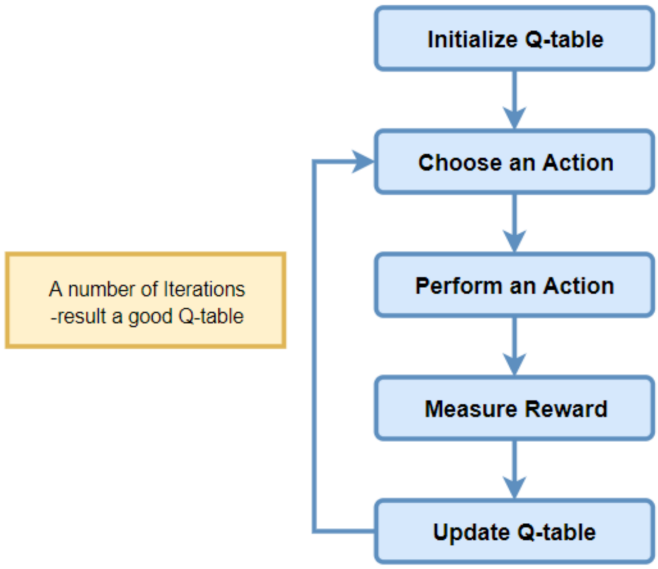

Figure 2: Q-learning Algorithm.

The Q-Learning algorithm is used in order to find the next best action, given a current state, seeks to learn a policy that maximizes the total reward.

3 Solution of problem

The solution involves the robot first being trained on a type of arena with particular characteristics by performing several iterations (epochs) and then tested on different environments to see if it has learned to generalize.

3.1 Robot features

The robot is equipped with a sensor called **footbot_base_ground** which allows to detect if the robot is above a black area or not. The sensor reads the color of the floor, has a list of 8 values which can be 0 (black) or 1 (white), for example if they are all at 0 it means that the robot is entirely on a black area.

3.2 States and actions

The base ground sensor has 8 values that correspond to 8 possible directions.

1. N : North
2. NW : North West
3. W : West
4. SW : South West
5. S : South
6. SE : South east
7. E : East
8. NE : North east

```
sensor_direction_names = {"N", "NW", "W", "SW",
"S", "SE", "E", "NE"}
```

Each value can be 0 (black) or 1 (white) this results in a total of states

$$2^8 = 256$$

.

There are three actions the robot can perform: moving forwards, to the left and to the right by 45 degrees. These actions are applied by giving speed to the wheels.

```
velocity_direction_names = {"WNW", "N", "ENE"}
velocity_directions = {
    ["WNW"] = math.pi / 4, -- 45 degree
    ["N"] = 0,
    ["ENE"] = - math.pi / 4, -- -45 degree
}
```

In this way the Q-table has a total of 256 (states) * 3 (actions) = 768 values are then applied to wheels velocity.

3.3 Reward function

The reward function is one of the most important aspect for building a Q-learning model. The reward function has been designed considering how many values of the base ground sensor are white (value=1) this counter is rappresent in the `white_sensors` variable. The rewards are computed basing of `white_sensors` counter and they goes from 0 to 1. The 0 value stands for no reward, the case when the robot is entirely on a black area, the 1 value rappresent the case when the robot stands entirely on a white area, where it receive

the maximum reward. The values between 0 and 1 rappresent the case when robot has from 1 to 7 sensor in black area. The following code rappresent the reward mechanism.

```
function get_reward()
 -- Rewards goes from 0 to 1
 local white_sensors = 0

 for i = 1, #robot.base_ground do
   if robot.base_ground[i].value == 1 then
     white_sensors = white_sensors + 1
   end
 end
  -- rw=(number of sensor in white area / total sensor)^2
 return math.pow(white_sensors / #robot.base_ground, 2)
end
```

As you can see from the code the reward value is calculated based on the number of sensors with value 1 (in the white area of the arena) divided by the total number of sensors(`#robot.base_ground`) squared.

$$(white\_sensors/\#robot.base\_ground)^2$$

3.4 Get state

To compute the next state, the 8 values of the base ground sensor are used in this way: The value of the state goes from 1 to 256 and the default value is 1 which indicates that the robot is entirely on a black area, the value 256 indicates that the robot is entirely on white area of arena. otherwise the value between 256 and 1 indicates that some sensor are on the white area and some other on the black area. The snippet below rappresent the code to calculate the next state.

```
function get_state()
  -- State goes from 1 to 256.
  -- 1 equals that all sensors are 0 black area.
  local new_state = 1

  for i = 1, #robot.base_ground do
    -- if the values of base ground is equals to 1
    -- there is white area under the robot
    if robot.base_ground[i].value == 1 then
      new_state = new_state + math.pow(2,i-1)
    end
  end

  return new_state
end
```

4 Training

The entire training phase was based on this 5 steps algorithm:

1. Get state based on base ground sensor.
2. Update the Q-table using the alpha and gamma hyperparameters, the current state of the robot, the action, the reward and the future state. Alpha is the learning rate, which indicates how much the old Q values are taken into account in the update. If alpha is very high, the robot's previous knowledge is almost completely ignored else if alpha is small, past experience is given a lot of weight.
3. Get the best action based on the state and k which is a hyperparameter that tells how strong the selection favours actions with high Q table value.
4. Perform the action previously obtained.
5. Repeat previous steps.

The following snippet formalize this algorithmin Lua.

```
--- This function is executed at each time step.
-- It must contain the logic of your controller.
function step()
        n_steps = n_steps + 1
  -- update counter
        if n_steps % MOVE_STEPS == 0 then
    -- Update
    state = get_state()
    local reward = get_reward()
```

```
    Q_table = Qlearning.update_Q_table(alpha, gamma, old_state,
    action, reward, state, Q_table)

    -- Perform the best action and update state
    action = Qlearning.get_weighted_action(k, old_state, Q_table)
    perform_action(action)
    old_state = state

  end
end
```

Initially I had tried to train the robot on the following arena.

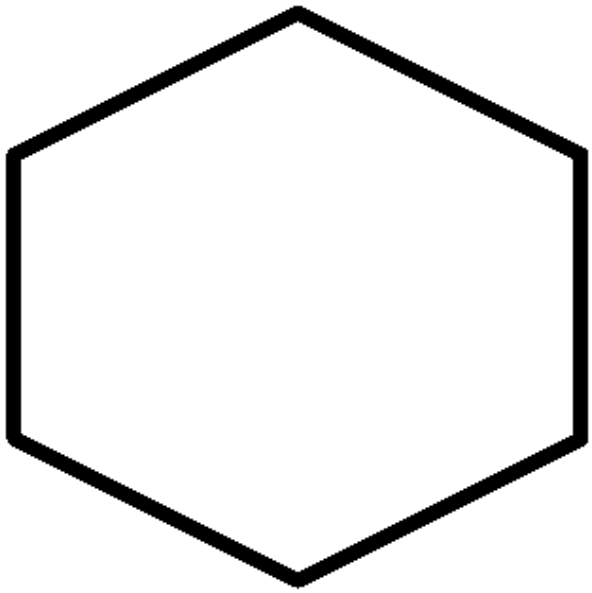

Figure 3: small-hexagon arena

With start position in center, performance was really good on it but when i tested the model on other arenas i observed that: In the arena with a large presence of white it is very easy to make high scores, but if you look at the running model you will notice that when the robot meets a black area of the arena of little thickness tends to cross it. So I decided to improve my training by trying an arena with very thick black areas like the following. This allowed

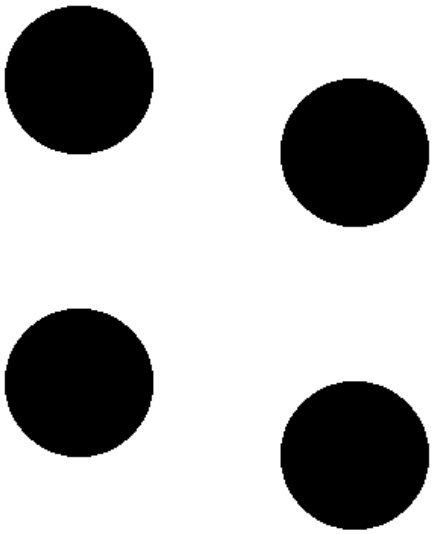

Figure 4: four spots arena

the robot to learn to stay in the black area for as little time as possible, trying to turn direction when it comes into contact with it, avoiding crossing it. After this clear improvement, I decided to train the robot in the following arena with thick black areas but with more characteristics than the previous one.

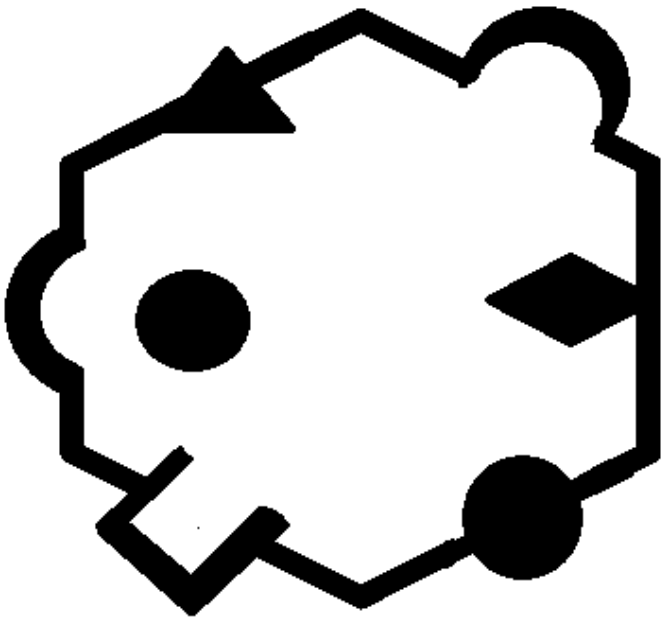

Figure 5: complex hexagon arena

This last arena turned out to be the best for training, with it the robot learned to stay as little time as possible in the black areas and when it meets small turns immediately to return to the white area. Further details of the results are described in the following chapters.

5 Testing

The testing phase use what has been learned in the learning phase. Mainly the robot has two competences:

1. competence0 is used when the robot is completely in a white area of the arena, it will make the robot move randomly.

2. competence1 is used when one of the values of the base ground sensors becomes 0 (black) then the robot uses an action to consciously change its state.

The follow code show how the competence1 works.

```
function competence1()
    ...
  local state = get_state()
  local action = Qlearning.get_best_action(state, Q_table)
  local subsumption = true

-- entirely on white part,
-- move random  if the state is equal to 256
-- if there is at least one sensor in black area
-- it skip this condition
  if state == number_of_states then
    subsumption = false
  end

  return subsumption, action_to_velocity(action)
end
```

Competence0 and competence1 are used to move robot in step() function as the snippet below show.

```
--- This function is executed at each time step.
-- It must contain the logic of your controller.
function step()
        n_steps = n_steps + 1

  -- Perform action
  if n_steps % MOVE_STEPS == 0 then

    left_v0_random, right_v0_random = competence0()
    subsumption1, left_v1_action, right_v1_action = competence1()

    --- Is minimum one of the sensors black?
    if (subsumption1) then
      robot.wheels.set_velocity(left_v1_action, right_v1_action)
    else
    --- if state is  ==256 all sensors are on the white area
    --- -> move random
      robot.wheels.set_velocity(left_v0_random, right_v0_random)
    end
        end
end
```

In the testing phase, the robot is tested in different arenas with different black and white parts by starting it in random positions of the arena or in the middle. We measure the performance of the score in the test phase using a metric that ensures we quantify how long the robot stays in the white areas of

the arena compared to the total area covered on the arena.

6 Results

Numerous tests have been performed and the robot has demonstrated a robust behaviour in most cases and environments. The metric used to assess how much time the robot spends on the white areas consists of counting how much time remains on the white areas out of the total number of counts as described below.

```
metric = math.floor(on_white_circuit_acquisition
                    /total_state_acquisition)
```

I think this is not an absolute correct metric because in arenas where the white area is much greater than the black area it is very likely that the score is high; conversely if the arena is predominantly black the score is purely low. In addition, in cases where the robot was started from a random position if it happens right inside a black area, the robot will necessarily make a lower score. After these considerations I decided to base my results not only on the score obtained but on the correctness of the robot's own behavior. As previously described I chose for the training an arena with black areas very marked so that the robot learned to turn at their contact and to cross them only in particular situations such as the presence of an angle that makes inconvenient for the robot turn around 180 degrees with sensors in the black area rather than crossing it because as already positioned in the right direction. The results obtained will be shown later, but take attention to the considerations made to give it the right relevance.

The complex hexagon arena shown above was used for the training phase while different arenas were used for the test phase. For training, the robot is positioned in the centre of the arena and also positioned in random position while in the test phase it is positioned randomly and different types of arenas are used. Initially, the robot was tested under optimal conditions without noise while in a second part the robot was tested by stressing the robustness of the behaviour by adding 0.5 noise.

6.1 hyperparams

At the beginning the robot is trained and tested without the presence of any noise on the sensors and actuators. The robot is trained for 100 epochs using the following hyperparameters:

Hyperparams	α	0.1
	γ	0.9
	k	2
	epochs	100

6.2 Train results

The results of the robot average rewards in the various learning experiences in the various arenas are shown below.

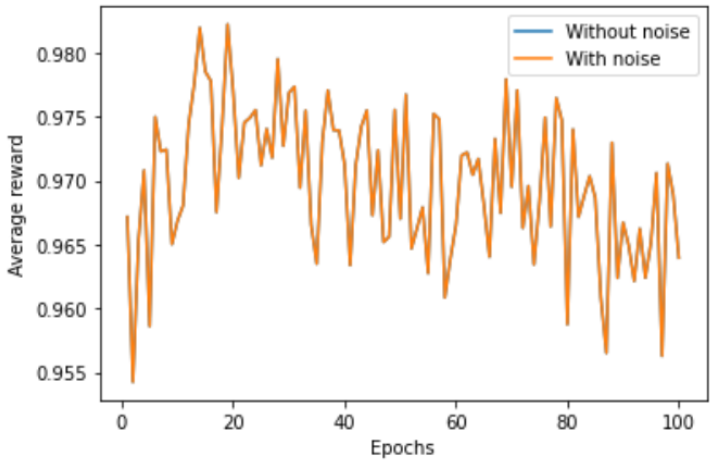

Figure 6: Train in small hexagon arena

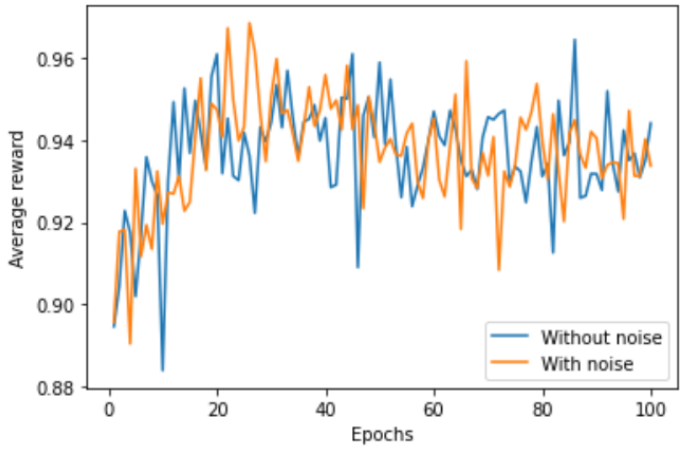

Figure 7: Train in four spot arena

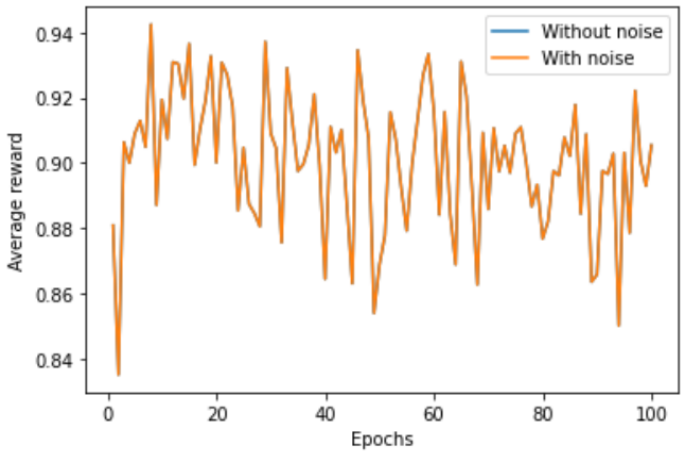

Figure 8: Train complex hexagon arena

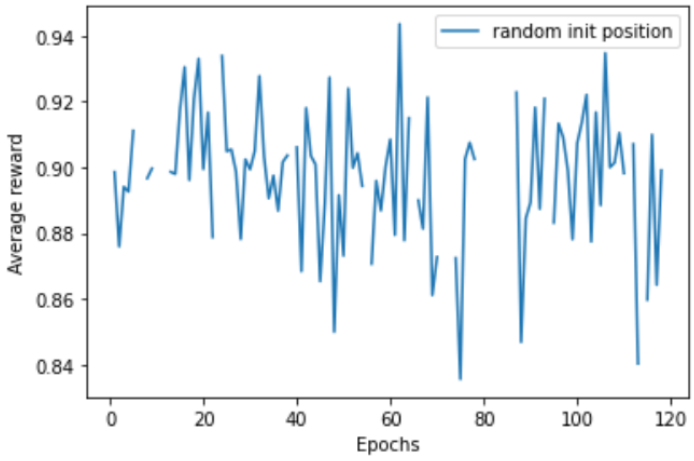

Figure 9: Train complex hexagon arena starting from random position

Initially, the rewards are lower, but in any case the trend is not linear or always increasing over time. To check the robustness of the behaviour, the robot was subsequently trained by adding noise on the sensors and actuators. The value of the hyperparameters is the same as above. Considering a noise case on sensors and actuators of 0.5. As we could see in some circuits, the presence of noise does not affect the average rewards, but as we shall see, it produces a slightly worse result in testing. While starting the robot in a random position still produces good average rewards but for the metrics used produces more fluctuating results, as can be seen below.

6.3 Test results

the results of the tests performed both in training arenas and in more complex arenas such as snake to random2 are shown below.

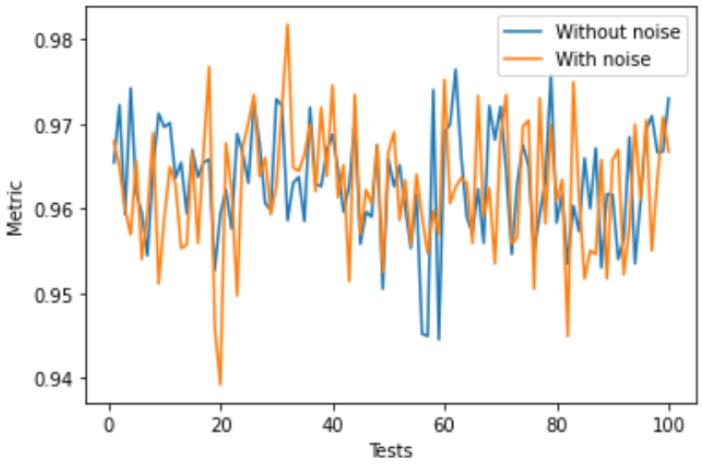

Figure 10: Test in small hexagon arena

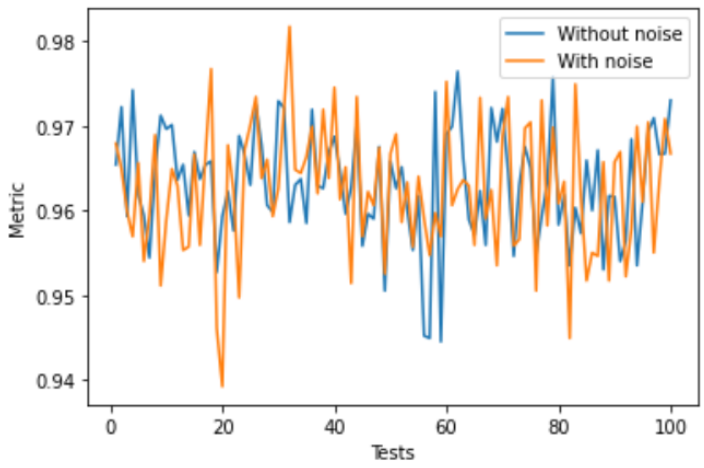

Figure 11: Test in four spots arena

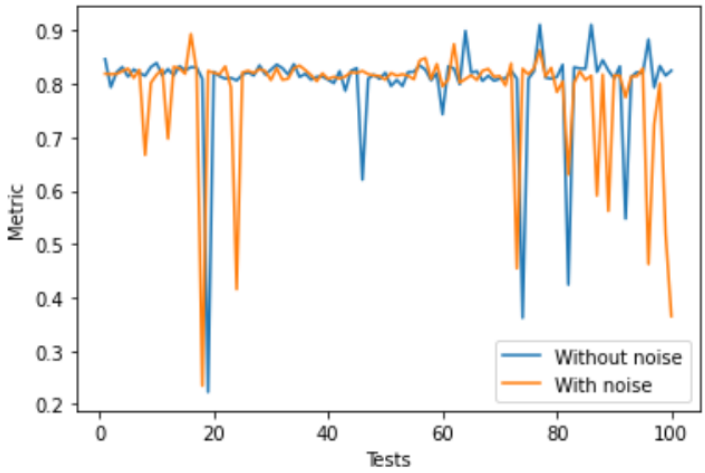

Figure 12: Test in complex hexagon arena

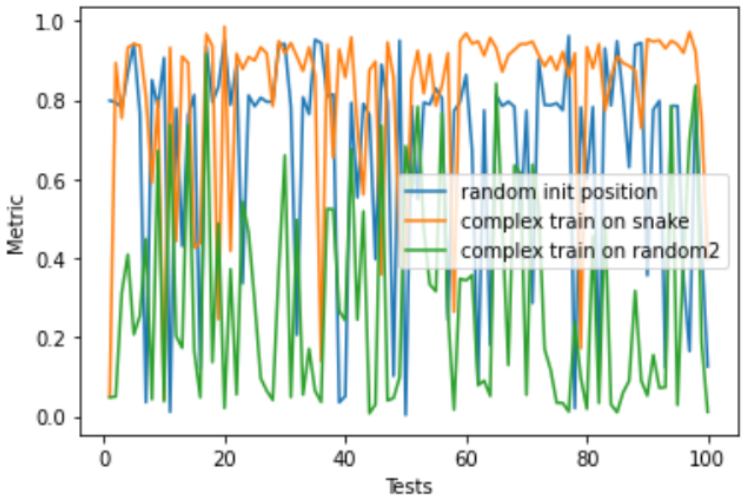

Figure 13: Test in snake, random2 and complex hexagon arena using start random position

The metric graph quantifies how much the robot stays entirely on the white areas (tested on the same arena used for learning). Increasing the number of epochs, the robot tends to make less mistakes by choosing the best action, even if the behavior as mentioned is not always incremental and situations may arise in which the robot makes more effort.

6.4 final considerations

As said before, this metrics doesn't reflect totally the behaviur of the robot, in fact observing more times the robot in test phase we can look that robot work well. Very complicated arenas make the correct interpretation of this metric difficult. In random2 the result is very low but observing the robot's behaviour in the simulator one can see that it actually moves discretely to stay as little time as possible in the black arena.

7 Conclusions

In conclusion, the project was very interesting, I think that reinforcement learning and Q-Learning are really important aspects in robotics field. I think that the project could be extended, by combining different behaviors always to avoid the black areas and to improve performance. I am very satisfied with the results obtained in the test simulations, as the robot as soon as it enters a black area tries to get out of it by rotating, not only does it stay there for the shortest possible time, but it also manages to stay within a perimeter and only get out under special circumstances.