Basketball bot

Q-Learning & Genetic Algorithm

By Group 19: Gal Rosenblum, Aviv Ohayon, Chaya Lasry and Joel Schreiber

Contents

1 Problem Description

The problem we chose to solve is of a robot that learns to pick up the ball and throw the ball into the basket. This problem consists of two subproblems: pick up the ball and throw the ball into the basket.

For the first sub problem: robot picks up the ball, we use the Q-Learning which we learned in this course and implemented in exercise 4. For the second sub problem: robot throws the ball into the basket, we choose to solve with Genetic Algorithm, and we also implemented a throw heuristic that tries to learn from the shot what actions should be taken to have a higher chance of shooting.

1.1 How the game work

We built all the graphics of the game using the $pygame$ and $pymunk$ packages. There are 3 types of objects in the game: STATIC objects that do not move in the game like the limits of the game, KINEMATIC objects that have all their moves done by the computer and DYNAMIC objects which are affected by physical data given to objects and space.

The object in the game:

- Robot - has arms, hands, and wheels. The robot can move left or right and up or down the arms or the hands, when the robot is close enough with it's hands to the ball it can also pick up the ball, and if the robot is already picking up the ball it can throw the ball with a certain power at the angles of the hands and arms. The robot is a kinematic object that receives command from the computer (lines code in the project).
- Ball - a dynamic object such that it is affected by the gravity of the space of the game and will stop his velocity.
- Basket - a static object the ball collides with.

During the run of the game the default run is learning of the robot to pick up the ball and throw the ball into the basket using the genetic algorithm. There is an option to replace the genetic algorithm with the learning heuristic we implemented and there is another option to run only the genetic algorithm, in this case the robot throws the ball into the basket from all possible positions to shooting.

There are several stages in the game, so that in each stage the robot has different possible actions.

The stages:

- The robot's distance from the ball is over 60 pixels.
- The robot's distance from the ball is less or equal to 60 pixels.
- The robot's hands distance from the ball is close enough to pick up the ball.
- The robot picks up the ball and moves to the shooting position
- The robot grabs the ball and tries to throw it.
- The ball is thrown.

2 Approaches to the solution

2.1 Q-Learning

In exercise 4 we implemented the Q-Learning Agent class and we saw how the crawler learned to crawl by using the Q-Learning agent. So, we took inspiration from the crawler and by using this agent made the robot learn to move towards the ball and when the robot is close enough pick up the ball.

The Q-learning does a generalized value iteration process, that means for each iteration the agent is update the $Q(s,a)$ as follows:

$$Q(s_t,a_t) = Q(s_t,a_t) + \left[r_t + \max_a\{Q(s_{t+1},a)\} - Q(s_t,a_t)\right]$$

Such that s_t and a_t are the state and action in iteration t, $Q(s_t,a_t)$ is the q-value of the state and the action in iteration t, α is the learning rate factor and γ is the discount factor.

The action the q-learning agent given to the robot to do is in chance epsilon is a random action from the legal actions, else given random action from all the actions with the highest q-value in state of the robot.

In terms of our implementation of functions that support the learning process of the robot, in the Robot class we have 3 functions that are used by the q-learning agent:

- getCurrentState(self) - return the current state that is different for each stage of the robot. (The stages we described are in the game description)
- getPossibleActions(self, state) - get state and by the stage of the state knowing which actions to return.
- doAction(self, action) - get action, perform the action it received and return the next state and reward.

An explanation for each situation, what is the state, the legal actions and the rewards that returned:

- The robot's distance from the ball is over 60 pixels

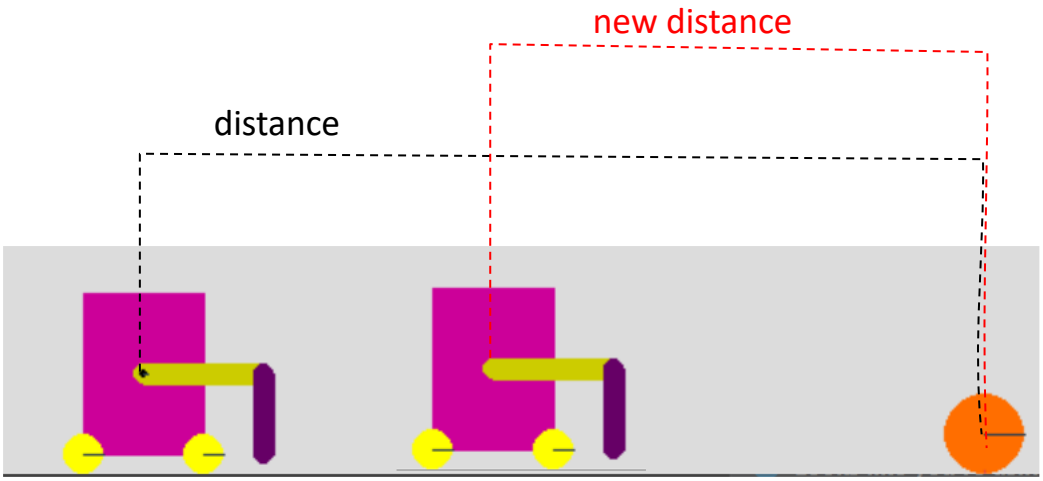


Figure 1

In this case the possible actions that returned are left, right, hands up, hands down, arms up, arms down. The state that is returned is the distance of the robot from the ball and an indicated parameter stage that indicates which stage the robot is in, we choose to return only the distance because the position of the hands and arms does not matter in the current stage of the robot, and also because later on if the robot and the ball are in different positions then the robot has already learned how to reach the ball and that way it is more dynamic and it will not count as a new state. The reward that is returned is $[distance\ -\ new\_distance]$, it means -10 to move left, 10 to right and 0 to hands and arms actions, because we want the robot to go to the ball.

- The robot's distance from the ball is less or equal to 60 pixels

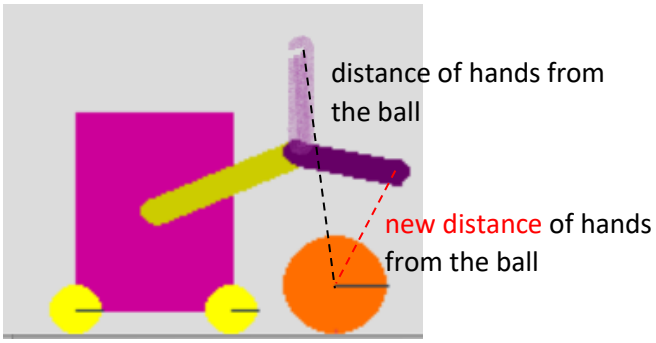


Figure 2

In this case the possible actions are all the possible actions in the previous stage except move right, such that the robot will not be able to collide with the ball. The state that is returned is the distance of the robot from the ball, angles of hands and arms (to calculate the distance of the hands from the ball) and stage of the robot. The reward is equal to $[distance\ -\ new\_distance]$ now is the distance of the hands from the ball when the action is one of the hands or arms actions, if the action is left the reward equal to -100, because we prefer the hands to be close to the ball.

- The robot's hands distance from the ball is close enough to pick up the ball

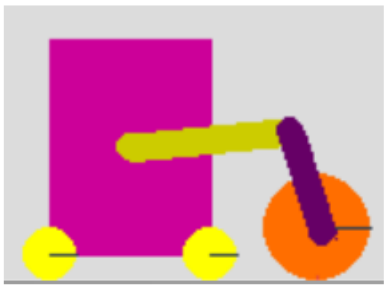


Figure 3

In this case the possible actions are all the possible actions in the previous stage and are added to the pick-up move. The state that returns is the same state from the previous stage and the reward is 10 to pick up action and -10 to others, clearly, we prefer to encourage the pick-up move.

- The robot picks up the ball and move to the shooting position

The robot can move only left or right (not both) until it reaches the shooting position. The reason there is a shooting position that the robot needs to reach, is because we want the robot to learn from that position (that will consider the same state) to throw the ball into the basket until it succeeds and then randomly choose a new point.

- The robot grabs the ball and tries to throw it

  We do not use learning in this stage because the process of learning is very slow, but still the functions we mentioned also support this stage. The possible actions are the hands and arms actions, increase the power and throw. After the running of the genetic algorithm the robot receives a vector of parameters of good angles and power that make the robot shoot the ball into the basket. When the action is one of the hands or arms action the reward is equal to the $[prev\ angle\ - new\ angle]$ when the pervious angle is the angle between the good angle the robot receives, and the angle of the robot hands or arms and the new angle is the angle between the good angle and the new angle of the robot hands or arms. For power up action the reward is $[good\ power\ -\ prev\ power]$, when previous power is the power before the increment and for the throw action the reward is the minus the difference between all the parameters (angles and power) from the good parameters, such that the higher reward is 0 to throw away the same values. The state that returns is position, angels of hands and arms, power, and stage.

- The ball is thrown

  The robot waiting for the ball stops moving and tries to pick up the ball again.

2.2 Genetic Algorithm

The second sub problem of robot try to throw the ball into the basket we decided to implement with Genetic Algorithm because the q-learning agent takes hours to learn how to throw the ball into the basket and the reason is the training of the robot is very long and even if the robot succeed to throw the ball into the basket (after hours or days) to succeed the next try to shoot to the basket the epsilon needs to be tiny. So, the idea to solve in Genetic Algorithm is easier, faster and optimal (it promises that the robot will succeed in throwing the ball into the basket).

We implement GA Class that supports the running of the game using the genetic algorithm.

The pseudo code we took inspiration from is:

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
      new_population ← empty set
      for i = 1 to SIZE(population) do
          x ← RANDOM-SELECTION(population, FITNESS-FN)
          y ← RANDOM-SELECTION(population, FITNESS-FN)
          child ← REPRODUCE(x, y)
          if (small random probability) then child ← MUTATE(child)
          add child to new_population
      population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4

To adjust the running of the genetic algorithm, we created a step(self) function in the GA Class that is called by the Basketball Class and added the throw(self, state) function to the Robot Class. The throw function gets the state of angles of hands and arms and power, and the function makes the robot throw the ball with the angles and the power it received by the step function in the GA Class.

Pseudo code of our implementation how the genetic algorithm solves the second subproblem:

1. First grill n_pop states in the **Population** list (in the init of GA Class).
2. Throw the ball with the parameters in the states in population.
3. For each throw, update the fitness value of the state in the **Fitness Values** list.
4. After all the throws of the states in population the algorithm performs the parts of the **Selection**, **Crossover** and **Mutation** and makes a new **Population** list of states that have more chances to throw the ball into the basket.
5. Return to 2 until the robot successfully throws the ball into the basket, in this case the state is saved and the next time the robot is in the same position it will know which parameters of angles and power to throw the ball to get shooting.

Explanation of the steps of the genetic algorithm:

- **Population** - a list of k randomly generated states of angles of the robot hands and arms and the power of the throw.
- **Fitness Values** - a list of k fitness values. Let b be the basket position (point in the screen) and let p be the closest point of the ball from the basket on the throw of state in the population list. The fitness value of the throw is:

  $$-|p.x - b.x| + b.y - p.y$$

  We want the shots with minimal distance between the basket from the closest point on the x axis and the case where the ball is above the basket will get higher value.

- **Selection** - choose randomly states with preference for states with higher fitness value.
- **Crossover** - take two states that are chosen randomly in the selection, in chance 0.9 make crossover between those states, random an index (in our case 1 or 2) and create two new states that consist of the previous states parameters.

- **Mutation** - in chance 0.9 take random index of state parameter (angle hands or arms or power) and replace it with new random value.

An example of running the algorithm with the problem of throwing the ball into the basket:

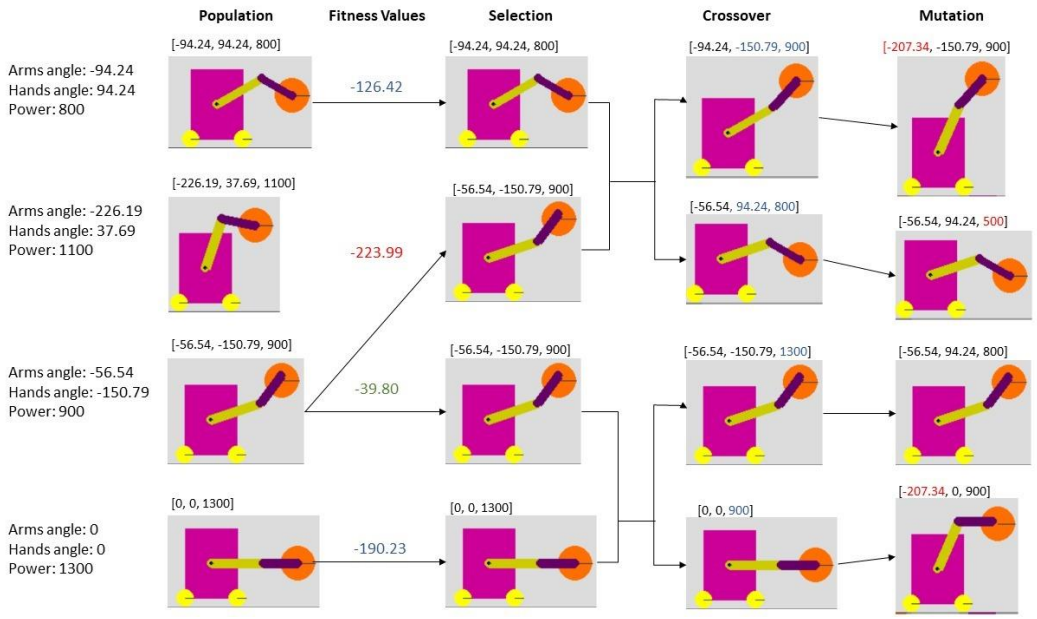


Figure 5

2.3 Throw Heuristic

The heuristic has a vector of numbers of the actions of the hands, arms and power of throw that is first chosen randomly and then in each throw the vector changes according to the data of the previous shot, the maximum point and the smallest distance between the ball and the basket. To prioritize cases of a ball above the basket and a smaller distance, we will raise and lower the weights of the vector, until we get that a robot throws the ball into the basket. We also added a probabilistic element that, with a chance of 0.6, will perform the action of lowering the hands or increasing the power, and the reason is we do not want to end up in loop with two bad shots, one above the basket and the other below, and each time we raise and lower the same weights.

3 Results and Conclusions

3.1 Q-Learning

It can be seen from $Figure\ 6$ that in each iteration the robot learns to pick up the ball and manages to achieve the first goal of the problem. In some iterations the number of steps is bigger than in previous iterations and the reason for this is also that the epsilon parameter is equal to 0.5 in the given graph, therefore sometimes actions that are not good are chosen randomly, which leads the robot to reach the ball in a position that it was not in in previous iterations and thus the robot is still in a training phase even in an advanced phase of the iterations (in the case there is a choice of raising hands and arms and the angles that the robot reaches the ball were not before).

We would prefer the epsilon to be 0.5 so that in new situations that the robot reaches, it won't be stuck for a long time on actions that exist with a low Q value and will also grill additional actions with a high value, such that the next time the robot knows how to choose them.

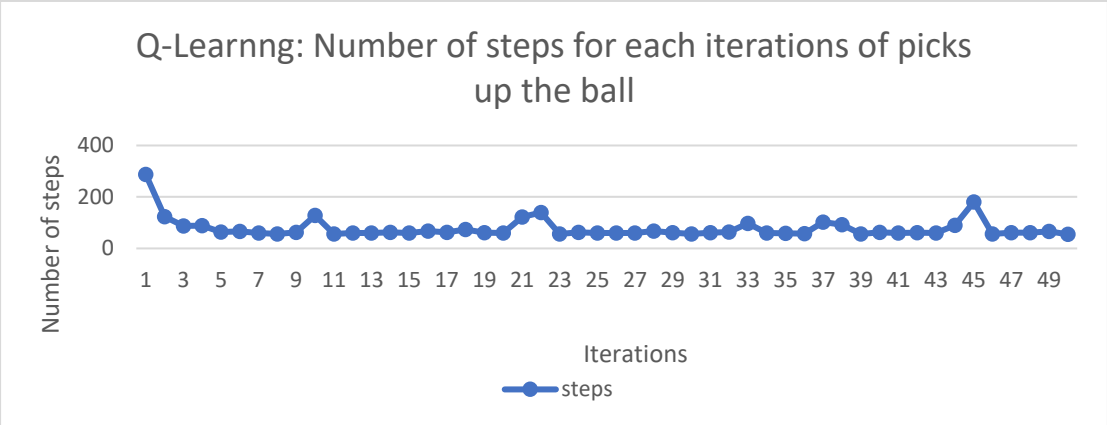


Figure 6

In terms of learning the robot for the second sub-problem of the project: throwing the ball into the basket, from the following graph in $Figure\ 7$ you can see in 217 iterations the minimum distance of the ball from the basket in each shot by using the genetic algorithm and the Q learning agent. After 12 attempts, the genetic algorithm manages to able the robot to shoot into the basket (and then the data is saved, so there will be a shot into the basket in every next iteration) and the distances of the basket from the ball are by a margin smaller than the distances that the agent achieves.

From the point of view of the agent, the distances themselves are initially very large and this is because it is still in the robot's training phase and the agent releases the ball relatively quickly. You can still see a very slow trend of getting closer to the basket.

So clearly the genetic algorithm is better than the Q learning agent, because it manages to achieve the goal very quickly compared to the agent.

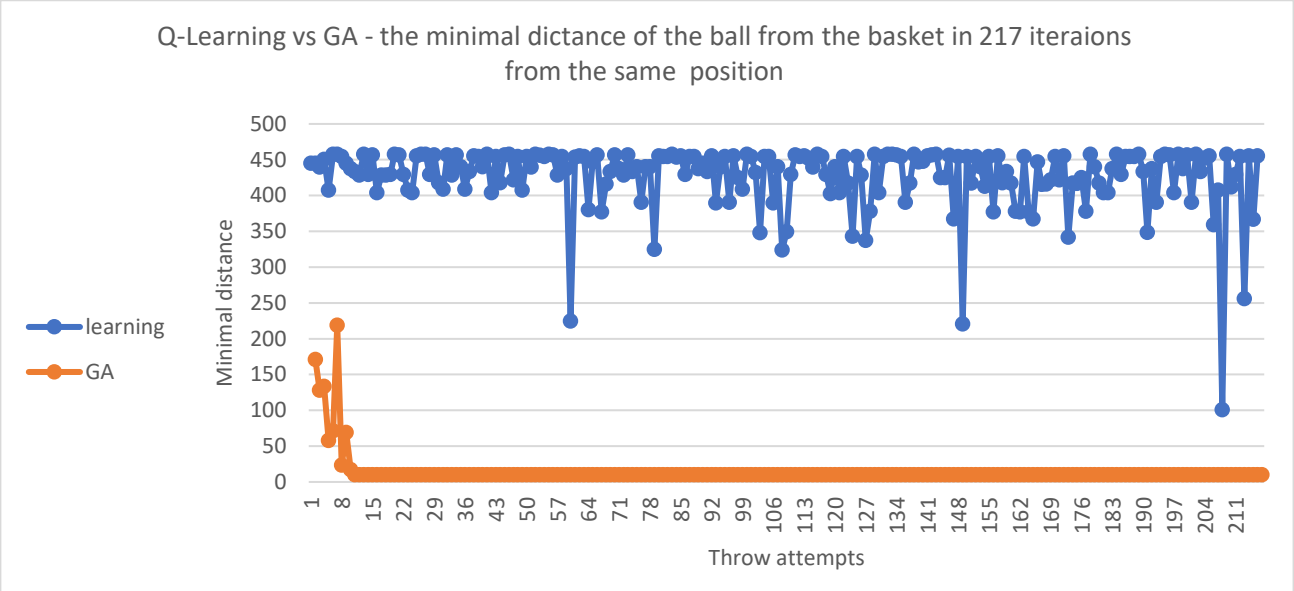


Figure 7

3.2 Genetic Algorithm

In the following graph in $Figure\ 8$ you can see the number of attempts of shots into the basket from each possible shot point using the genetic algorithm.

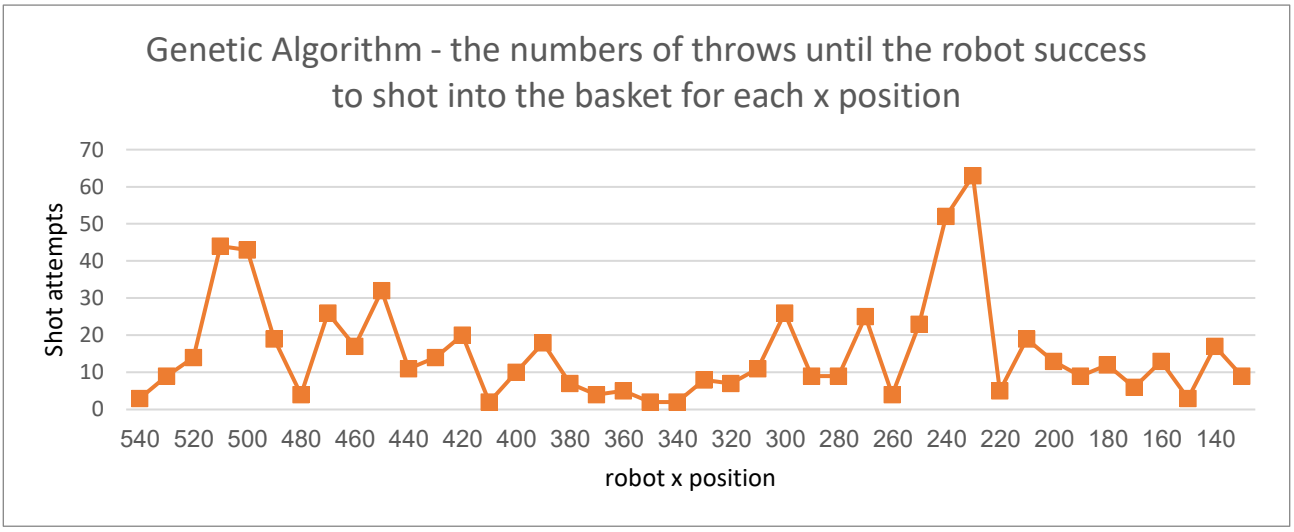


Figure 8

From the data we received it can be concluded that on average it takes 15.46 attempts for the robot to score the ball into the basket. That is, the genetic algorithm succeeds in improving the population already in the second generation, often achieving a shot in the basket.

We added a comparison (shown in $Figure\ 9$) between runs of the genetic algorithm with population sizes of 10 and 4 for each possible shot point and the number of at

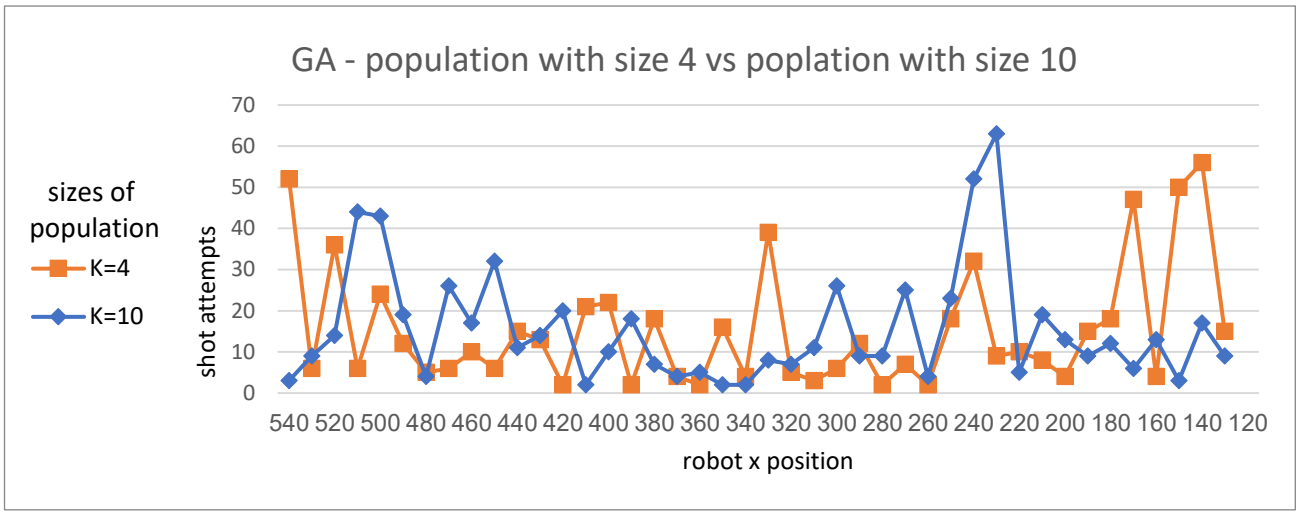
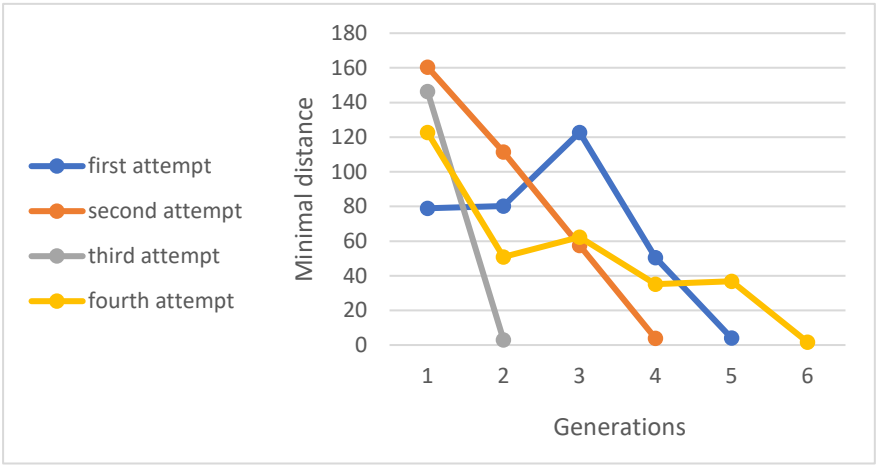


Figure 9

tempts by the robot until it managed to put the ball in the basket.

We obtained that the robot takes an average of 15.46 attempts until it succeeds in throwing the ball into the basket with a population of size 10 and with a population of size 4 it takes an average of 15.51 attempts. So, the size of the population does not matter that much because on average we will achieve the same results relatively quickly.

The following graph in $Figure\ 10$ shows 4 different attempts of throwing the ball into the basket using the genetic algorithm and the minimum distance between each generation in the throwing attempts (each generation is the size of the population which in our case is 10 states). It can be concluded from the graph that in each generation there is progress (relatively because sometimes fewer good values are drawn in the part of the mutation and crossover) and thus the minimum distance decreases. As we mentioned before, the average is 15.46 attempts of throwing a ball to achieve the goal and in other words it takes on average 2 generations to succeed in throwing to the basket, so, a case of 6 generations in shot attempts is considered an extreme case, but still exists sometimes.



10 Figure

3.3 Throw Heuristic

We run the throw heuristic for each possible shot point on the screen, the heuristic usually works really well succeeding in throwing the ball after a number in the range of 20 shot attempts, but there are cases when the heuristic enters a loop of bad actions, we tried to solve it by making a change certain in the vector of heuristics with a probability of 0.6 and thus there is a greater chance for different shots with a greater chance of entering. Still sometimes enters a loop of bad shots that repeat themselves as you can see in the next graph that for $x = 200$ or $x = 180$ it takes more attempts until success.

On average, it takes the robot 16.21 attempts to throw the ball into the basket until it succeeds.

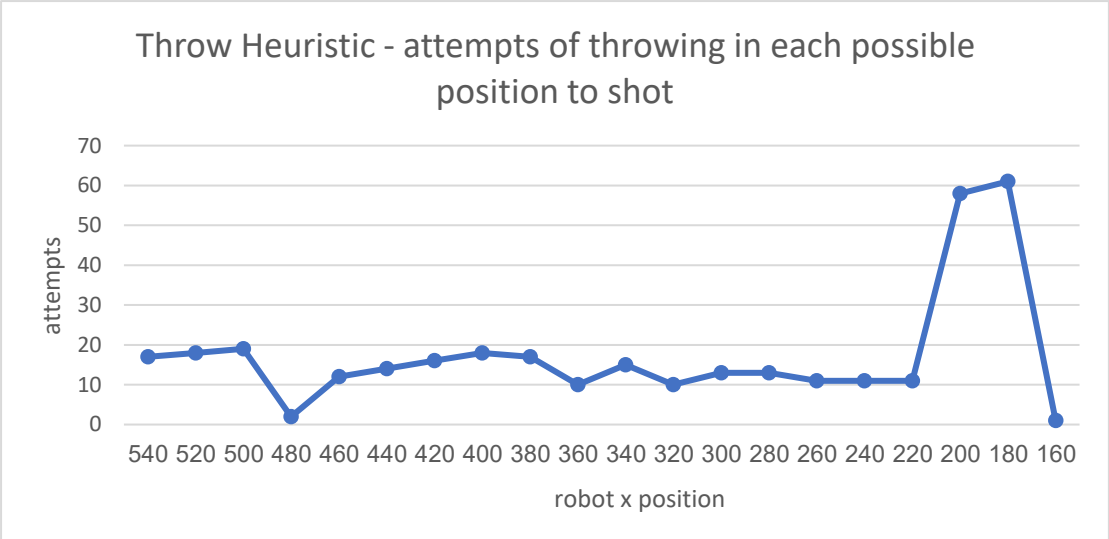


Figure 11

Comparing the throw heuristic and the genetic algorithm (in $Fogure\ 12$) it can be seen that in most cases the genetic algorithm achieves better results, and its average is also lower than the heuristic average.

In addition, the genetic algorithm is optimal and inevitably manages to shoot the ball into the basket, and in some cases the heuristics do not always achieve the goal, and this is also the main reason we implemented the genetic algorithm class.

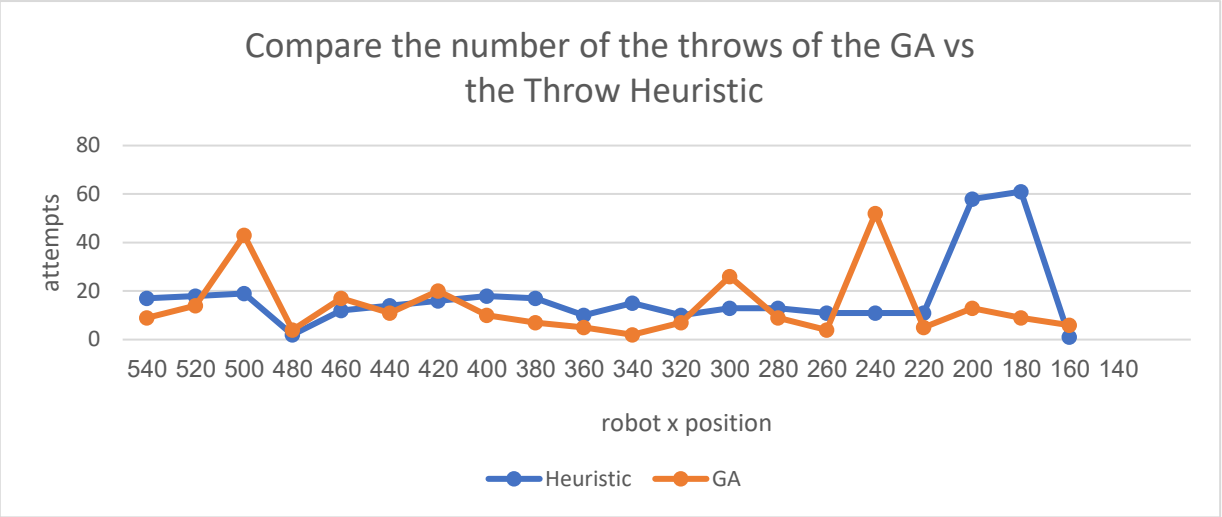


Figure 12

4 Discussion

To sum up, from the conclusions we presented, the Q-learning agent solves the first sub-problem we presented very quickly and make the robot success to pick up the ball with less steps from the previous iteration.

In the second sub-problem of the project, robot try to throw the ball into the basket, there are three approaches we proposed to solve it, the best approach is the Genetic Algorithm and learning the robot to pick up the ball is very slow and inefficient, a heuristic that we implement gives good results most of the times, but there are still extreme cases, and therefore the Genetic Algorithm is better than it.

5 How to run the project

1. Before running the project, install all the packages in **requirements.txt**
2. Enter the following command in the command line:

   - python3 basketball.py

3. The default throwing approach is the Genetic Algorithm. To switch between the Genetic Algorithm and the Throwing Heuristic press 't' on the keyboard.
4. There is an option to run only the genetic algorithm for any possible position for a shot on the screen. At the end of the run, the robot repeats the shots that entered the basket and then returns to the last shot approach defined.
5. To start or stop a run of only the genetic algorithm you must press 'g' on the keyboard.
6. It is possible to change the learning factors and the step delay by pressing the buttons on the dashboard on the screen. (Marked in the next figure)

Figure 13