

**Artificial Intelligence**

**Final Report**

Made by :

Hang Liu

Adrian Golęba

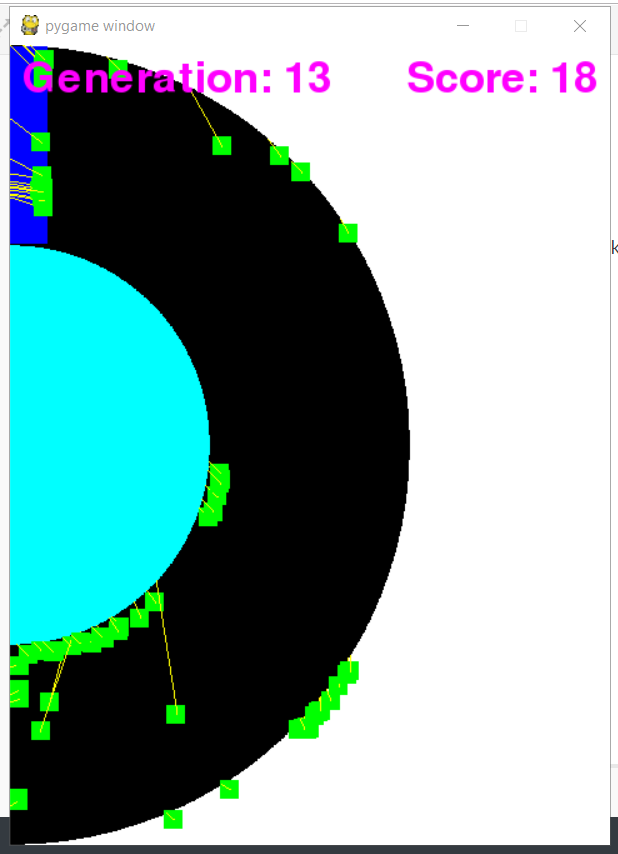
We implemented the game part by the library of pygame. The game model is simplified by two half circles. A big circle around a small circle. And the center of the circles are both in the center of the left part screen edge. The cars start from the bottom part of the screen. And the final lane in the top part of the screen, at the end of the cycle. If the car crosses the final lane, we regard this as a success, otherwise, if the car runs out the time limit, or collapses the border, the car fails. The car will feel the distance by 5 sensors which are embedded in front of the car. The car utilizes these sensors to feel the distance to the left and right border, and the distance ahead of it. This mechanism mimics the human being when they are driving a car, and also to ensure the car will not cheat. The car will not know any prior information about the cycle. It will explore by itself. Because calculating distance is implemented by pure math, we don’t have an advanced human eye or laser detecting system, nor do we find a good map drawing system. So we decided to use simple circles instead of complex map components. This will let us easily calculate the distances using math. And about this part, the code is in myItem.py library.

We did our project by 2 methods. First one is **genetic algorithms** **and neural networks**(controlAI.py). And the second one is **reinforcement learning by using a policy gradient method**(controlRL.py) and also a **neural network**.

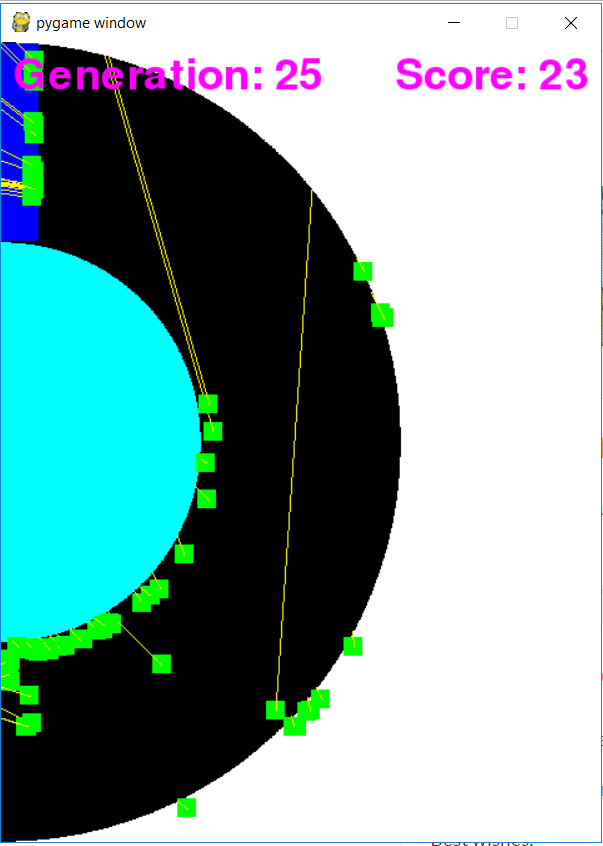
As for the genetic algorithm and neural network, we designed the neural network by 5, 7, 2 architecture. 5 inputs are each sensor readings, 7 hidden layers and 2 outputs. The 2 output will return an 01 array, could be [00], [01], [11], [10]. And it means turn right with decreasing, turn right with increasing, turn left with increasing and turn left with decreasing. Because the cycle is circular, the driver must turn their wheel all the time by common sense, which means we don't want them to go straight. We train this model by using gameStart() function as the only one in main function. It will initialize an initial car number by carnum, in our case it is suggested to be 50 to 100. And 60 rounds for training. We used sigmoid function as a filter function. Because it will not be interrupted either by extremely great value, nor by extremely small value. And in each round, we save one best performance AI by their weights and bias.

And at last we optimize the genes of the best AI by genetic algorithm. We randomly disturb the parameters of the AI by changing the scale or adding minusing the bias and weight parameter. The car will feel much better in order to achieve our goal in 15 to 20 generations.

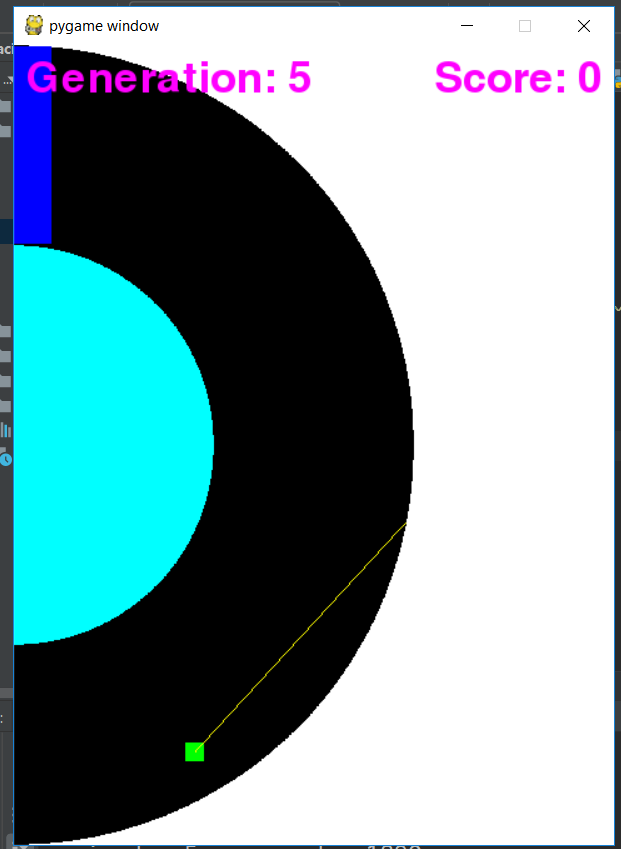
In the reinforcement learning part, the most important part is how to design a good reward function. And unfortunately, we do not have a pretty good method except for guessing and trails and errors. And this part game will start from gameStart\_by\_RL() function. If we put this into the main function, it will run. The good thing is the output of policy gradient can output a continuous value. And this algorithm will get and update the information(observation values, action values, rewards values) or data every round. It will store every round information by store\_transition() function. By calculating the probabilities of each action and according to the probability, we choose the action. About the learning, it is based on the reward function. We use \_discount\_and\_norm\_rewards() function to make rewards de-attenuation. We need to wait about 500 to 700 generations to feel the car is making sense on moving in order to achieve our goal.



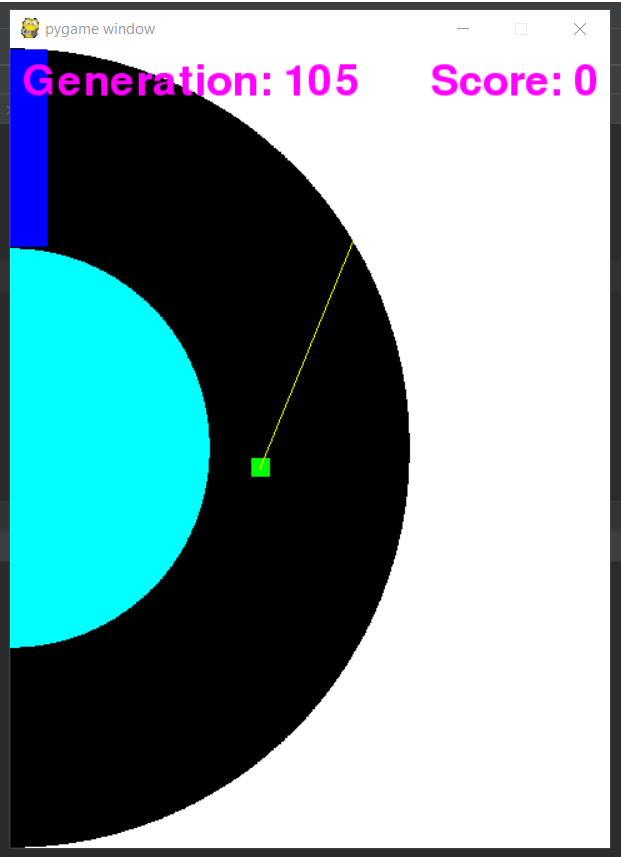
**Screenshot from genetic algorithm at generation 13**



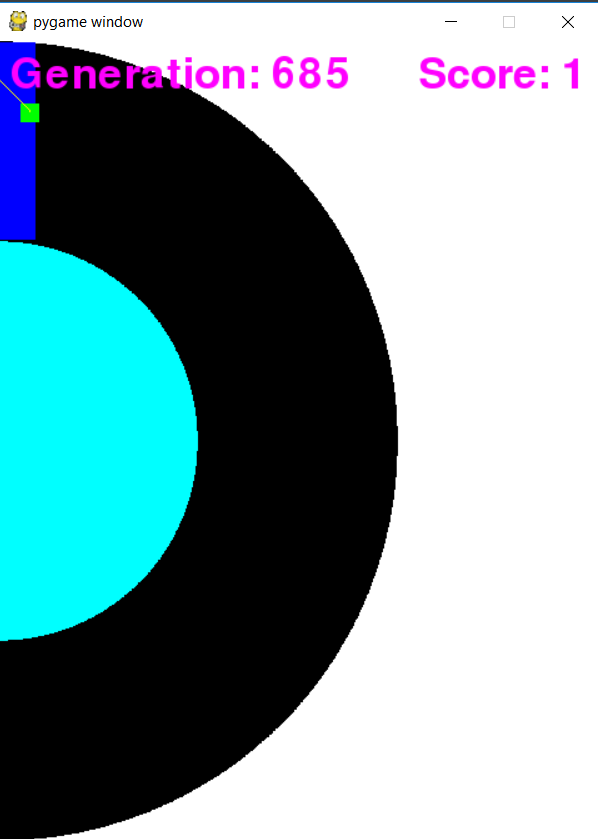
**Screenshot from genetic algorithm at generation 25**



**Screenshot from reinforcement learning at generation 5**



**Screenshot from reinforcement learning at generation 105**



**Screenshot from reinforcement learning at generation 685**

So in each generation of genetic algorithms, it usually has 100 cars in our experiment, and it needs around 15 to 30 generations to have a reasonable move among cars. And compared to reinforcement learning,

It runs 1 single car at a time as 1 generation, and it needs about 500 to 700 generations to see there is a clear effort on this. But we think if we could have a better reward function, it could significantly increase the iteration time of reinforcement learning.