COMP3211
Fundamentals of Artificial Intelligence
Final Project

Imitation of ELEC1100 cart on track
Rex Chan 20353506
The Problem

In ELEC1100, students are required to design a cart to cover a track using logic gates and wires. This task tries to recreate the problem but using artificial intelligence to solve it.

The initial thought was to create an environment with a track, and a cart with complex controls, with both wheels being able to choose from reverse full speed, reverse half speed, stop, half speed forward, and full speed forward. The two wheel design soon was discovered too complex and was scraped, leaving a simpler design with actions: [forward, backwards, rotate left/right by 15 degrees, and going left/right in an arc]. This proved to be too complex for tabular Q learning, so at last a design with actions: [forward, backwards, left 15 degrees, right 15 degrees] was set.

The environment is made of a 600*600 grid (which proved to be too complex for Q learning in the end), with a table indicating whether the specified coordinates are part of the track.

Along the track, invisible baits (like those in pac man) are dropped to motivate the cart to move forward, with reward = 2, cumulative. (0:0, 1:2, 2:4, 3:6...) The area that is not the track would bounce back the cart. The final goal has a reward of 10000.

To prevent the program from getting stuck, a time constraint is set, with every bait refreshing the timer to their rewards and each action reducing it by 1. The game resets with reward = -1 when the timer becomes 0.
Solving the Problem

At first, simple tabular Q learning taught in class is used. However, with a problem with this many states, it took a very large amount of time to learn, with every trial ending up with a cart circling at the starting point.

Needing an algorithm that is more suitable for more complex problems, I chose Deep Q Learning. With the addition of a neural network, learning is significantly faster.

In the beginning, there were no baits nor timers, only negative rewards that kills; this caused the cart to only spin non-stop at the starting point. This is because of the imbalance of positive to negative rewards, demotivation the cart from moving at all as moving brings negative rewards only.

Baits are added so that the positives would balance out the negatives, and it increased the efficiency of learning.

Eventually, negative rewards that kill didn't seem to be any good, as it resets the game every time the cart goes the wrong way, making it hard for the cart to learn what's further into the track. So it is changed to a border that bounces back the cart and still gave a -1 reward. Eventually the negative reward got removed as it demotivated the cart from moving.

Finally I have a realization: why not make it simpler by giving ‘eyes' to the cart? Before, the only observation given to the cart is its own position. And only counting the ones that are reachable, there are 40000 possibilities. But now, with eyes seeing if the ground ahead is walk-able, there are only 8 possibilities, and so the learning rate skyrockets.

Also, there were originally 3 dead-ends, but the cart couldn't distinguish which way is correct, so I removed it.
Environment setup and Parameter tuning

I used Python 3.6.3 on a Mac to setup the program. Using Tensorflow, pandas and NumPy, data handling is a breeze.

As for model tuning, I tried many different setups, the basic 4 commands (forward, backward, turn right, turn left), 6 commands (adding in arc left and right), and none have successfully conquered the maze yet, but the most consistent one seems to be the 4 commands simple one.
Conclusion

When I first learnt about tabular Q Learning in class I thought it was already really smart, and this project really blew my mind with all the other splendid algorithms.

Tabular Q Learning is definitely very weak, and is only suitable for small-scoped problems.

DQL is a really interesting algorithm; it combines the idea of human brains and neural networks and uses it to make decisions in a computer. Making it way faster than Tabular Q Learning.

The problem might be a little too complicated for tabular Q learning and DQL, as neither have solved it yet (after literally thousands of iterations), and there is no time to mess around with other algorithms, but I look forward to using Dueling DQL or A3C to tackle this problem.

Also, group projects are hard, and even harder when done alone.
Reference

Morvan 莫煩
https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/