Introduction: The purpose of this project is simulate an abstract object that can freely move and maneuver around obstacles, such as streets and other objects. This deviates from our original Bitcoin prediction project because of the inaccuracies in properly predicting the fluctuation of Bitcoin prices. Rather than predicting prices of Bitcoin, this object will predict an optimal path from point A to point B with minimal errors. The object should be able to learn the environment as it moves and eventually efficiently maneuver around. Additionally, the object should be able to adapt to any changes made within the environment, and make decisions accordingly. Eventually, we would like to be able to implement this simulation with real vehicles. It will be a way to improve self driving vehicles and help them efficiently maneuver around the environment.

Background:

Path Finding Simulator for Mobile Robot Navigation:

A robot was implemented different path finding algorithms (Breadth-first search(BFS), Depth-first search(DFS), Bellman-Ford, Dijkstra's Algorithm, and A* (self made based on best-first search) to go from point A to B. BFS and DFS are both uninformed search, meaning that neither searches have no previous knowledge. Whereas Dijkstra's and Bellman-Ford Algorithm uses previous knowledge. The results for each algorithm was compared based on finding the shortest path from point A to point B. While the robots went about path finding, they also obtained fully and precise information of the route. Information such as location of the wall, the length of the distance before turning and such.

This project used different algorithms to find the best time. It can help the project since we are trying to minimize the amount of trials it take the object to complete the route/maze.

Machine Learning for Car Navigation

Neural networks and markov models were used to to learn driving patterns. Neural networks were for short term predictions whereas markov models was used for long term(driving event). Short term as in something that can be done by the human brain, a route someone knows already. Neural networks learn feedforward from previous events and recurring events. Driving event as in last minute changes in the usual route. If there was an accident and a U-turn is needed. Markov models are used for random changes; it learns from past and current events. However, it believes that future events can be predicted better with current events rather than past events.

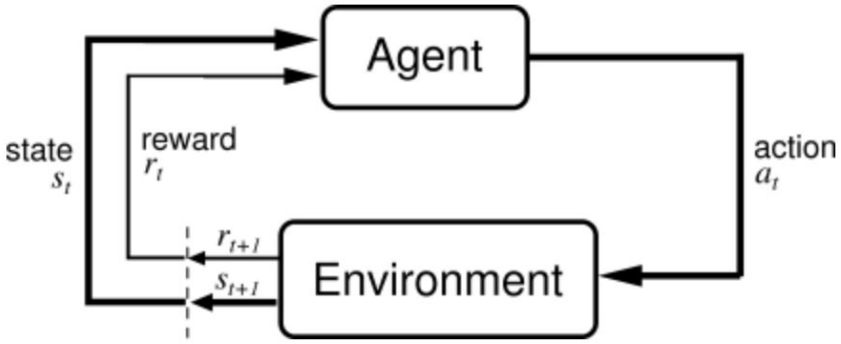
This idea would be perfect for those who have a set routine during the week (ex. Driving from home to work and vice versa). It did consider the two main different outcomes of self navigation.

Development:

1.Choosing the algorithm.

Reinforcement learning: The algorithm gets to choose an action in response to each data point. It is a common approach in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It's also a natural fit for Internet of Things applications. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was.

2. Planning/Drawing schematic of our algorithm.



Agent: Vehicle.
Environment: Surface with obstacles.
Actions:

- Moving: left, right, up, and down
- Detecting Obstacles.
- Maneuvering Obstacles.

State:

- The vehicle moving.
- Detecting an obstacles.
- Vehicle not moving

Rewards:

- String/number indicating the state is good or bad.

3 .Designing the Data Structures of our Objects

The following is pseudo code, we highly recommend to see the actual code to fully understand what the pseudo code means.

- Car
    - Function move {be able to move in more than just 4 directions}
    - Function Detecting {To detect approaching obstacles or walls }
    - Function Cerebro {To save the collected data such as traveling by itself(the heart of the algorithm)}
- Fix wall (this is actually kinda part of the environment)
    - Initiate with randomize wall paths, most likely we will have to implement an algorithm to make sure there is path from point a to point b.
    - Width, height, and area. We need this info so we can make it more realistic so walls do not overlap with each other.
- Location
    - Making an object of "location" can help us maximize retrieving locations of cars and walls.
- Environment
    - A canvas with a fix width and height

4. End goals at the end of development.

- Multiple cars be in sync to reach the same goal from a to b
    - There are going to be different paths from a to b
    - The cars can't overlap or push each other. Each car would have its own physical property. This means that a car would not be able to take the same path as another car.
    - Less trials
    - Shorter travel times
- The cerebro of each car to be in sync with other cars.
- Our walls need to generate random paths.

Prototype:

Please refer to folder "code/Simulation1"
You can download and run in linux with terminal command ./gradlew run
Or you can also used and IDE

Conclusion: From the prototype, we can see that the moving object took several attempts to full master the map. Eventually, we would like for the object to be able to move on curved roads and avoid moving obstacles, such as other cars. In future simulations, we would like implement more moving objects following the same map and learning the environments at the same time as the other objects. They can then communicate with each other to try and not crash into each other

and move accordingly. Furthermore, we would like to randomly generate maps to see if these objects can adapt to new randomized environments. It would be impossible to have actual cars crash into walls to predict the optimal path, so we would have vehicles calculate the path internally by sensing nearby walls. We hope that these additions would help improve self driving vehicles in the future.

References:

Path Finding Simulator for Mobile Robot Navigation:
https://www.researchgate.net/publication/254012352_Path_finding_simulator_for_mobile_robot_navigation

Machine Learning for Car Navigation
https://link.springer.com/chapter/10.1007/3-540-45517-5_74?no-access=true