Intro:

Currently I have implemented the project in an incremental fashion by applying RL on point masses in maze environments. This simulates a car (the point mass) and road (the maze) on a smaller scale which allows me to visualize and understand what the RL algorithms are doing before I apply them on a larger environment with more kinematic constraints. Each stage in the incremental development stage attacked a specific challenge before adding on more complex behaviors.

Stage 1:

In the first stage I implemented a random action point mass navigation in the PointMaze environment. This established my baseline understanding of the environment, observation space, action space, and reward structure. The reward structure used was a sparse reward structure (trying to mimic RRT in the random action aspect) in which a reward of 0 is awarded for any state where the agent's position was more than 0.45 units away and a reward of 1 when the agent is within 0.45 units of the goal. This binary reward structure provided the agent with no meaningful feedback during exploration and usually ended up with the agent either stuck in a corner, bouncing off walls, or randomly finding a way to the goal. However, we can't afford these errors when designing a reward structure for a car since.

Stage 2:

After analyzing the issues with stage 1, I implemented a distance-based heuristic that gives the agent more information. The distance-based heuristic calculates the vector between the position and the goal then applies a force in that direction. Applying this improved the navigation of the ball compared to stage 1. However, this method introduced two new issues: overshooting and difficulty in avoiding obstacles. The agent would overshoot the goal and start creating an oscillating behavior due to the momentum of the ball not being accounted for (a big force is applied on the ball and it doesn't have time to slow down before it gets to the goal). The other issue is that the ball was always trying to take a straight path to the goal which obviously won't work when obstacles are present.

Stage 3:

To solve the issues in stage 2, I implemented a velocity component (PD controller) to solve the oscillation problem. The PD controller would slow the ball down when it gets closer to the goal by applying an opposite force which is proportional to the velocity. This essentially adds a sort of braking system for the ball. Now in straight line paths, the ball is able to reach the goal perfectly without overshooting. To address the situations in which the ball is stuck or hits a wall, I implemented a checking strategy that looks into the past 10 iterations and detects if the ball has been stuck in one area for too long and then applies a force in a direction towards the goal node to nudge the ball in that direction. Although this solves the problem of a single point mass getting from start to goal, it still collides with the walls and we can't afford to do that when driving a car.

Stage 4:

In order to solve this issue I implemented Q-learning because the agent learns actions through experience. By discretizing the continuous state space and defining a set of discrete actions, our agent was able to discover more complex navigation strategies such as going around obstacles and avoiding wall collisions.

Stage 5:

The final stage of the initial phase of the project was extending the code to accommodate two point masses, each with different start and goal nodes. I implemented a custom environment wrapper that detected collisions between agents and applied a penalty when they collided. And through Q-learning, both agents tried to balance goal-seeking behavior with collision avoidance. However there were some issues with this reward structure since the balls would sometimes get stuck in a corner or a wall. This is an issue that I'm currently working on getting to work such that it can be applied in an environment with cars.

Next goals:

- Adapt and apply the learning algorithms to a car environment (conform to the car's kinematic constraints)
- Figure out how to better avoid collisions
- Analyze/learn how to use the CARLA simulator system (CARLA github)
- analyze/learn how to use the MACAD-Gym training platform (MACAD-Gym)