GROUP 36
Isaac Walter
Tyler Foster
Mak Phipps

Project 4
11/14/25

System Requirements

Our objective for this project is to implement and evaluate reinforcement learning algos to control a simulated race car, where it will have a starting point and a finish line on a predefined track and our algos will finish in the minimum number of steps. The car has 2 continuous variables being position and velocity but the agent can only control velocity or acceleration at each turn which will be from the values [+1, 0, -1] referring to the total velocity. The velocity variable will have a limited range of [-5,5] in either direction. We want our algos to be able to deal with non-deterministic heuristics because in this project there is a 20% chance that any chosen acceleration fails, meaning the accelerations will be equal to 0 for that step. We will have three environments (racetracks) to test our algos on, these tracks will use characters (S, F, ., #) to represent Start, Finish, Track, and Wall. We need to be able to detect that the vehicle is on the track or if they crash into a wall. This system will also require tuning parameters like learning rate and discount factor so running multiple experiments (at least 10 for each track) will help us find statistical evidence to decide the proper value of our parameters.

The algos that we will be using are Value Iteration, Q-learning, and SARSA and all will be implemented from scratch.

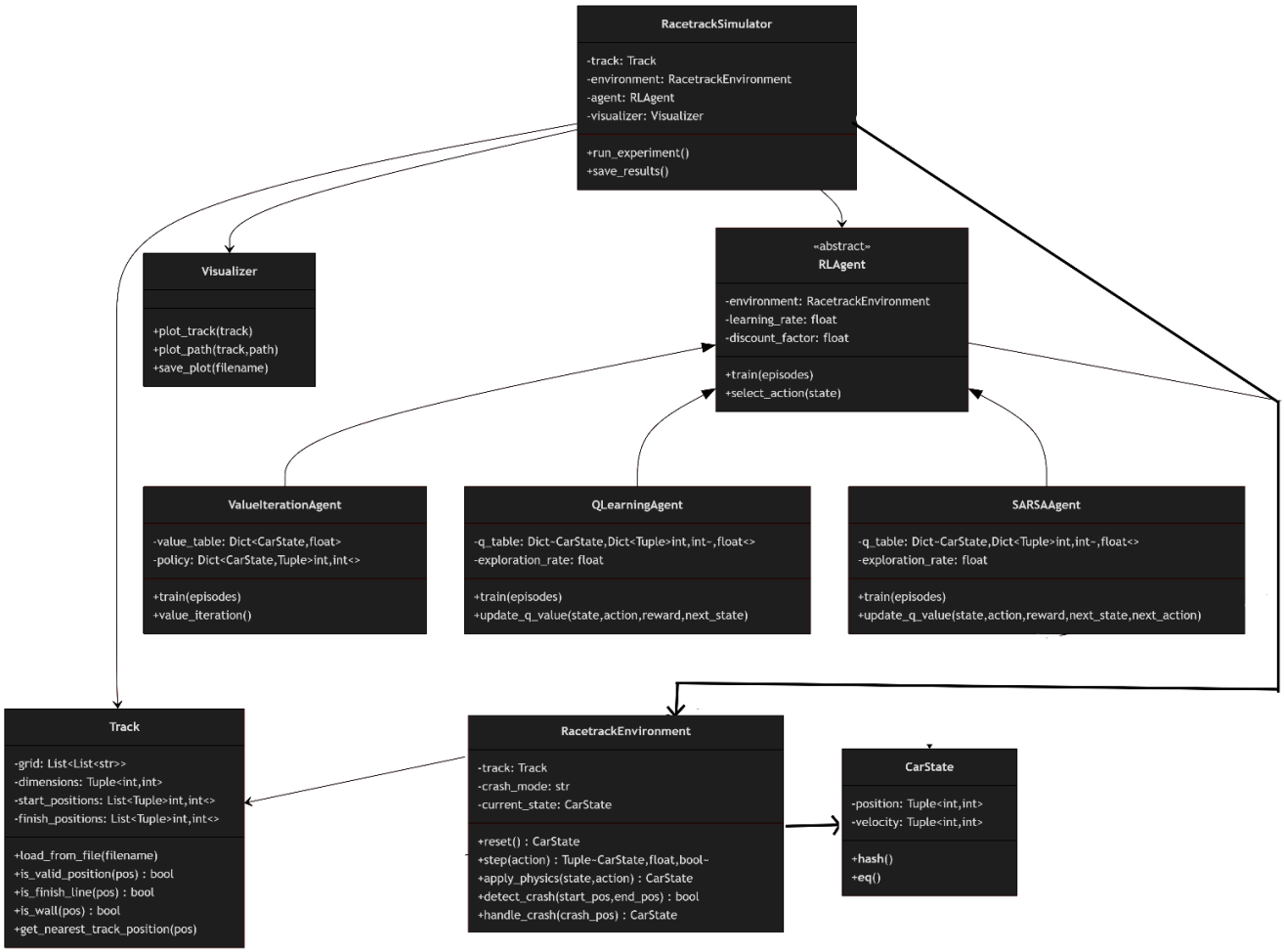
Value Iteration is a model-based dynamic programming algorithm that iteratively computes the optimal value function for each state. It uses the Bellman optimality equation to find the optimal policy by considering all possible state transitions and their probabilities. For the racetrack problem, Value Iteration will use our knowledge of the transition dynamics (80% success, 20% failure) to compute the theoretically optimal policy.

Q-learning is a model-free, off-policy temporal difference algorithm that learns action-value pairs without requiring a model of the environment. It updates Q-values based on the maximum expected future rewards, causing it to learn the optimal policy while following an exploratory behavior policy. Q-learning's off policy nature makes it good for finding optimal paths even when exploring.

SARSA (State-Action-Reward-State-Action) is a model-free, on-policy algorithm that learns the value of the policy being executed. Unlike Q-learning, it updates Q-values based on the actual action taken in the next state, making it more conservative and less risky, which may be beneficial in environments where crashes carry significant penalties.

Our program will output the code in a graphical plot of the cars path for a given run and will be saved as a png with a title following the format [GROUP_ID]_[ALGORITHM]_[TRACK_NAME]_[CRASH_POS]. Like every project there will be a cell with the parameters Group_Id, Algo, Track_Name, and Crash_Pos so we can easily run the desired algorithm on the desired track, where Crash_Pos will control the type of crashing allowed for the run. To expand on Crash_Pos there will be 2 possible states NRST and STRT where NRST means when we crash we will reset to the closest available track space and STRT means we will reset at the starting line.

System Architecture



^FIGURE 1

Figure 1 shows the class diagram for the system we plan to develop. The racetrack reinforcement learning system follows a modular architecture centered around the RactrackSimulator class which will be the main controller coordinating all the components. This simulator manages 4 core systems: track management, environment simulation, learning algorithms, and visualizing our data.

The track class forms the base of the system, and is responsible for loading and parsing ASCII track files that define the racetrack. It provides geometric operations including validating position, detecting finish lines, and finding nearest track positions for crash recovery.

The Racetrack Environment class will implement the simulation layer, managing the cars state transitions according to the velocity rules we were given for the project. It handles the core dynamics of velocity changes and acceleration limits ([-5,5]) and the 20% action of failure

probability. Crucially this class also implements the crash detection methods using Bresenhams line algo and the 2 crash recovery methods (NRST, STRT).

The learning system is built around an abstract RLAgent class that defines the common interface for all algos, with implementations for ValueIterationAgent, QLearingAgent, and SARSAAgent. Each agent will maintain its own specialized data structures: value tales for value iteration and Q-tables for the temporal difference methods. The architecture will support both model-based and model-free approaches while sharing common reinforcement learning parameters.

The visualizer class will handle the output generation required, showing the path of the car's learned trajectory from start to finish.

Data flows through the system in this pipeline: the Track loads configurations, the Environment manages state transitions and crash handling, the Agent learns optimal policies through interaction, and the visualizer generates performance outputs.

System Flow

1. Input handling
    a. Parameters cell will take GROUP_ID, ALGORITHM, TRACK_NAME, and CRASH_POS as inputs
    b. Inputs passed to RacetrackSimulator to select the specified algorithm, track, and crash rule.
2. Parsing
    a. ASCII text file will be parsed by the Track class
    b. Calculates and stores grid dimensions, as well as wall, start, and finish locations.
3. Apply Algorithms
    a. Apply from ALGORITHM parameter - either ValueIteration, QLearning, or SARSA.
    b. Apply crash logic - either NRST or STRT
    c. Implement Value Iteration
        i. Agent is provided full MDP model (transition probabilities and rewards) from RacetrackEnvironment
        ii. Iterates over entire state space
        iii. Extract final policy after the value function converges
    d. Q-Learning
        i. Not provided access to MDP model
        ii. Initialize Q[s][a] table
        iii. During each step, it uses an ε-greedy policy (mostly greedy, sometimes random) to choose an action and observe the next state and reward.
        iv. Finds the value of the optimal next action
        v. Updates the Q[s][a] based on that optimal value

    e. SARSA
        i. Not provided access to MDP model
        ii. Initialize Q[s][a] table
        iii. During each step, it uses an ε-greedy policy (mostly greedy, sometimes random) to choose an action
        iv. After observing the next state, it uses the same ε-greedy policy to choose the next action
        v. It updates Q[s][a] based on the value of the actual next action taken rather than the best possible action
4. Output
    a. A trial run using the final learned policy (run greedily, without exploration) is used to generate a path.
    b. The Visualizer uses the track grid to create a matplotlib plot
    c. Saves the plot as a PNG file with the proper convention [GROUP_ID]_[ALGORITHM]_[TRACK_NAME]_[CRASH_POS].png

Test Strategy

Our testing includes the following three algorithms: Value Iteration, Q-Learning, SARSA. Each of these algorithms will be tested on these three provided test tracks: 2-track, U-track, W-track. Every algorithm will be tested on each track 10 times where we will record the appropriate statistics so that we can accurately compare the three algorithms' performance.

Testing Objectives:

1. Confirmed tracks are parsed correctly and the ASCII maps match their expected grid layouts. This will require manually checking them for correct detection of the grid dimensions, start line positions, finish line positions, open track cells, and walls.
2. Confirm the environment simulation is accurate. The velocity updates, acceleration limits, non-deterministic acceleration, and crash detection will all be checked by observing the recorded data created from running an algorithm on a track.
3. We will confirm that the two different versions of crashes are the result of appropriate car movements. This requires manual confirmation that each crash type is the result of the correct car movements.
4. Each algorithm will be tested 10 times per track. This will result in 90 different runs. We will measure the following statistics and confirm that they are reasonably close to expected and realistic values:
    a. Track completion status
    b. Number of steps taken

    c. Number of crashes
    d. Number of learning iterations
    e. Exploration patterns
    f. Complete path taken
    g. Time to complete
5. Lastly, we will verify the track and path taken by the car are accurately presented in the output png file and that the file name is correct.

Overall, our testing plan checks for correctness and performance of the algorithms across the three tracks, allowing us to compare and visualize their performance.

Task assignments & Schedule

Task Assignments		
Task	**Description**	**Assigned**
Track Parsing	Read in the ASCII files so we can apply the algorithms to them.	Robert Phipps
Track Rules	Creating code that constrains the algorithms to the track so that we can accurately test algorithms. (Bresenham's Algorithm)	Tyler Foster
Value Iteration	Running the value iteration algorithm on at least 10 experiments for each track and generating data	Isaac Walter
Q-Learning	Running the Q-Learning algorithm on at least 10 experiments for each track and generating data	Robert Phipps
SARSA	Running the SARSA algorithm on at least 10 experiments for each track and generating data	Tyler Foster
Output plot as PNG	Saving the plot from the visualizer as a PNG so we can visualize the path	Isaac Walter
Testing	Testing	Everyone
Documentation	Recording our test results and creating a report that summarizes our results	Everyone

Communication plan

GitHub	All code will be shared between group members on a github repo for the best version control
Google Docs	Group members will collaborate on a google document to share notes, test results, and share thoughts
Discord, email, text	Group members will use Discord, email, and texting for quick communication

Schedule

Task	**11/24**	**11/25**	**11/26**	**11/27**	**11/28**	**11/29**	**11/30**	**12/1**	**12/2**	**12/3**
Track Parsing	Not st…									
Track Rules		Not started								
Value Iteration			Not started							
Q-Learning				Not started						
SARSA					Not started					
Output plot as PNG						Not started				
Testing		Not started								
Documentation		Not started								