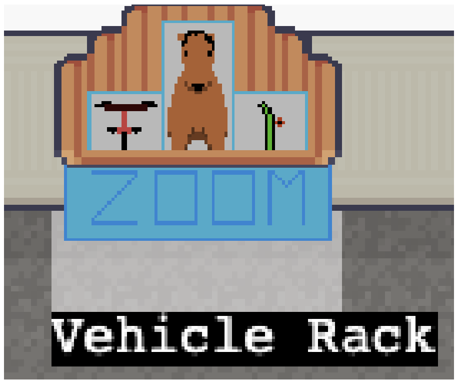
CS4530 Group 402 Project Report

Abhay Bisht, Ethan van Heerden, Liam Evans, Sam Phillippo

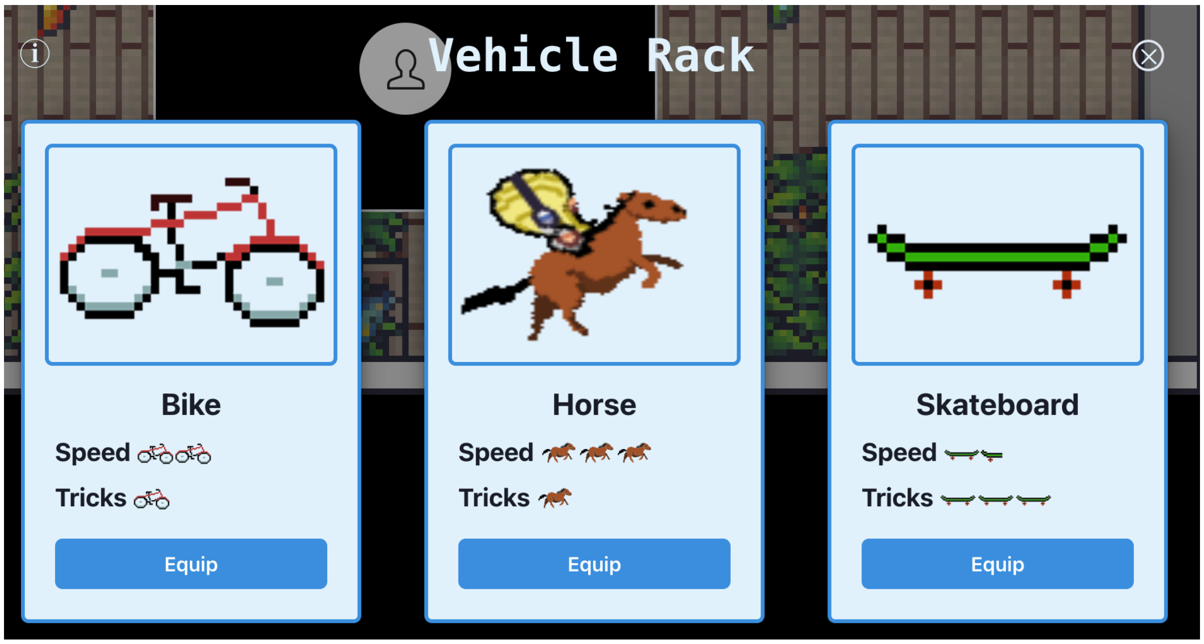
Feature Overview and User Manual

Our new feature for Covey.Town was the addition of vehicles. Players are able to select either a skateboard, bike, or horse to roam around the map faster, as well as participate in a new word-typing minigame as a means to do tricks on their equipped vehicle. In order to build our project, download our code as a zip file (can be found here), unzip it, and run `npm install` to install our project dependencies. Then, in the `townService` directory, run the command `npm start` to start the backend, and in another shell window, run the command `npm run dev` in the `frontend` directory to start the frontend. After these processes finish, open the localhost link to load into the town.

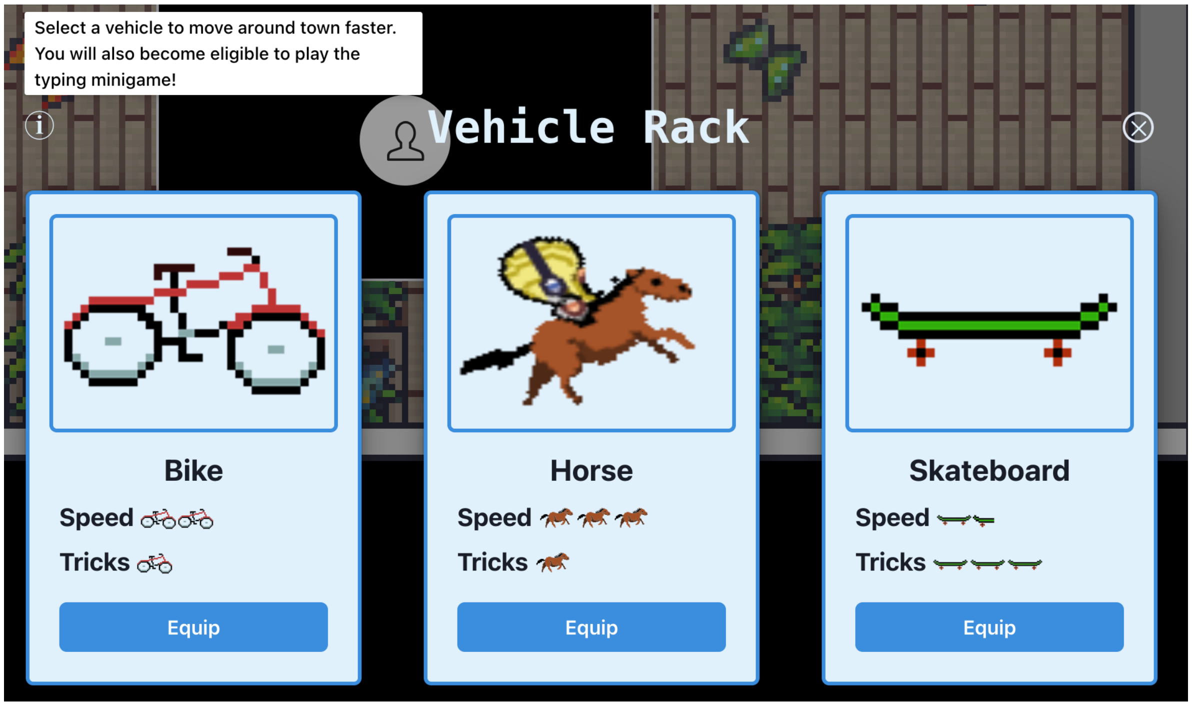
From here, you can see your character loaded into Covey.Town. At the top of the map relative to the spawn point, there is an interactable area with the title "Vehicle Rack":



Once in this area, press the spacebar to open up the vehicle rack modal:

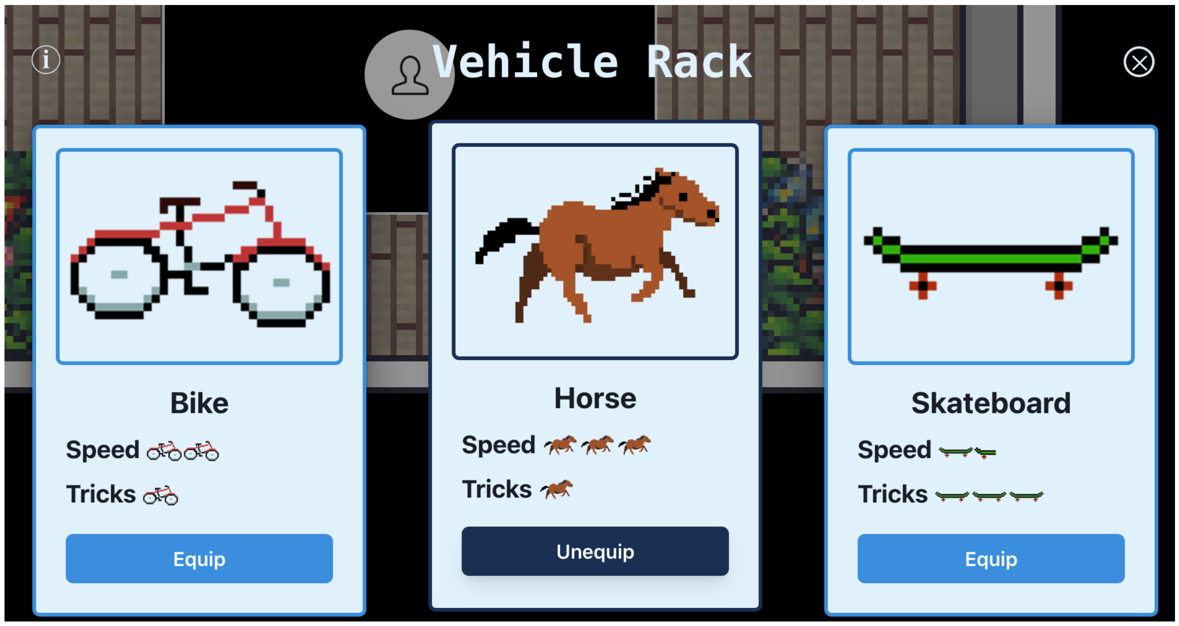


You can also hover over the info icon above for a small tutorial message:

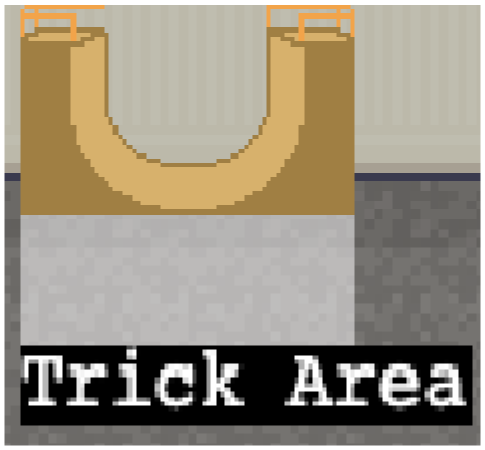

Select a vehicle to move around town faster. You will also become eligible to play the typing minigame!

Click on the appropriate "Equip" button for your desired vehicle to equip that vehicle. After you dismiss the modal, you should see your player using the new vehicle. You will then be able to transverse the map faster at different speeds depending on what vehicle you have:

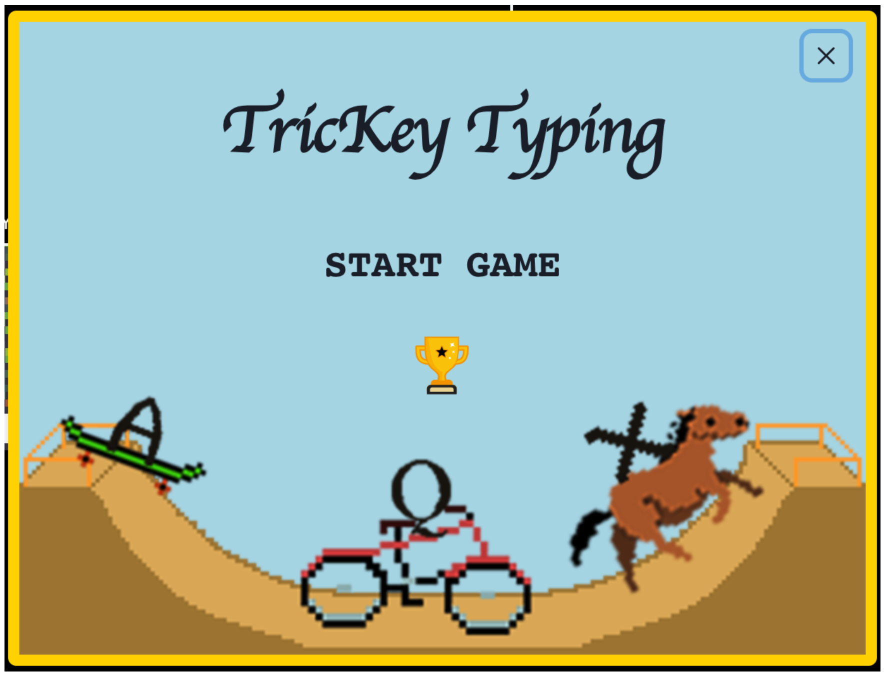
To unequip a vehicle, return to the vehicle rack and press the "Unequip" button under the vehicle you currently have equipped.

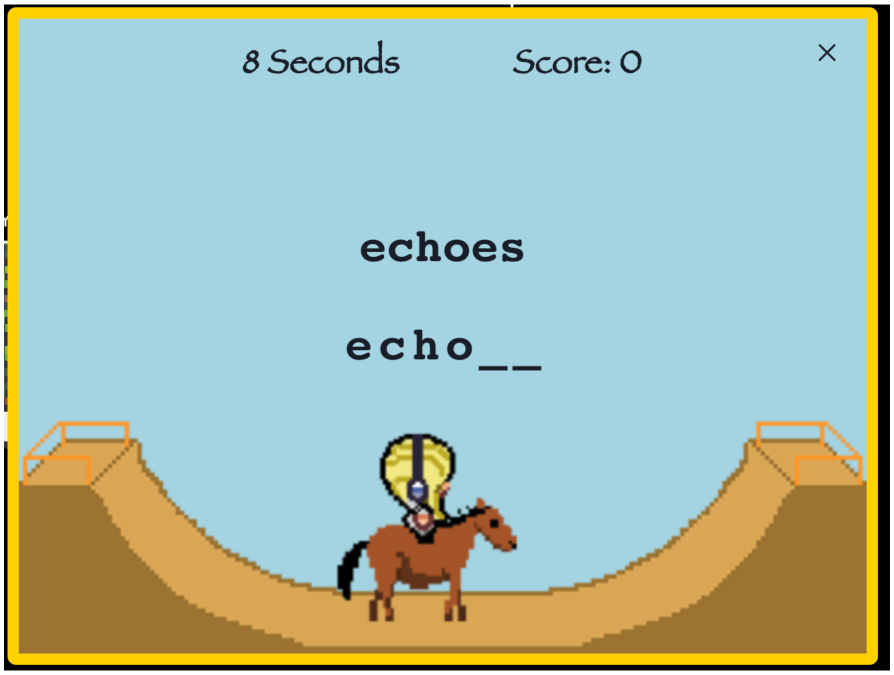


In order to play the trick minigame, ensure the player has a vehicle equipped, and then move the player to the interactable area titled “Trick Area” (located immediately to the right of the vehicle rack):

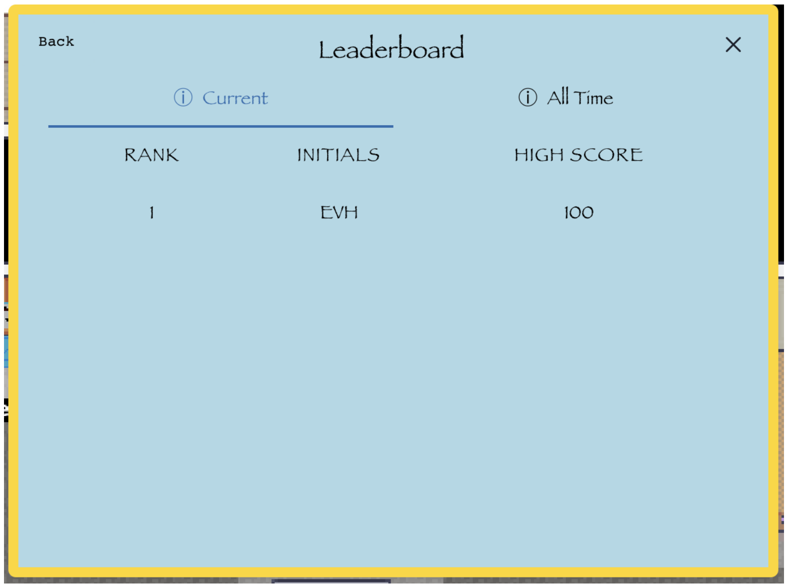


Once in this area, press the spacebar to open up the game modal. Press “Start Game” to start the game, and type in words as they are presented to score points. After each correct word you type, you should see your player complete a trick on their vehicle.

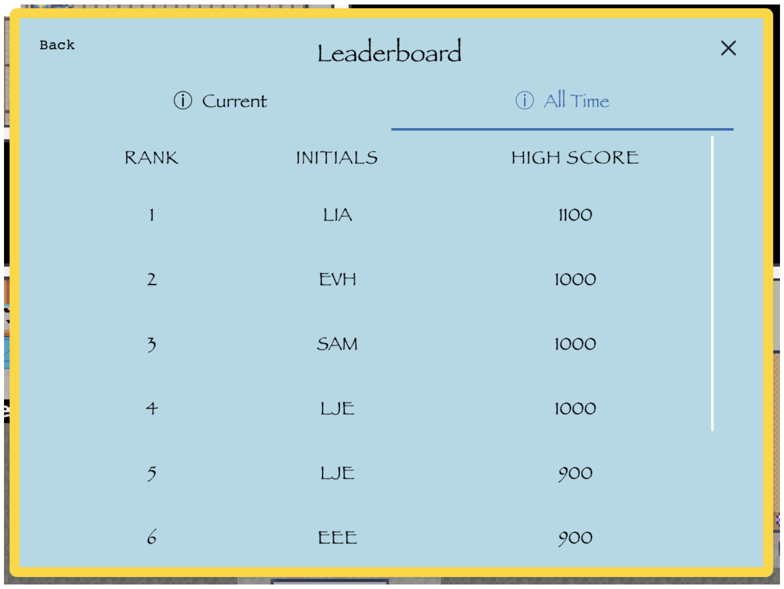




When time runs out, you will be prompted to enter 3-letter initials to save your score, and then you will be brought back to the game homepage. From the homepage, select the trophy icon to view the leaderboard:



RANK	INITIALS	HIGH SCORE
1	EVH	100



RANK	INITIALS	HIGH SCORE
1	LIA	1100
2	EVH	1000
3	SAM	1000
4	LJE	1000
5	LJE	900
6	EEE	900

There is both a “Current” tab to view the leaderboard for all players in the current town session, as well as an “All Time” tab to view the top scores of all time in all towns.

You can find our deployed app here.

Technical Overview

Our project consists of three main sections: the vehicle implementation, the vehicle rack implementation where players can equip and unequip vehicles, and our trick mini game.

For our vehicle implementation, we initially had a Vehicle superclass and a subclass for each vehicle, all of which was stored in the backend. Each vehicle subclass simply stored its speed multiplier and its vehicle type (as a string), and then we had getters for each value. However, due to some import issues, we decided to move our Vehicle superclass into the shared types file and then implement concrete instances of our superclass within the PlayerController file. This also meant that we added associated 'vehicle' fields within the Player and PlayerController files, which simplified our implementation and made it easier for the players to equip and unequip vehicles. This was the right design because it follows OOD principles since in the real world, players can have vehicles, so the player classes should have vehicle fields as well.

In order to stay consistent with the look and feel of Covey.Town, we needed to create custom sprites and animations based on the player's vehicle. After designing and creating our custom animations, we converted them to sprite atlases which we were able to load into TownGameScene. In order to use these new atlases, some abstraction was required, like renaming "misa" references to "walk", and creating a new "createMovementAnimations" function. Additional abstractions were made to the PlayerController to use different animations/atlases based on the vehicle property. This was the right design because it abstracted the logic for sprites and movement animations which would make adding future vehicles easier.

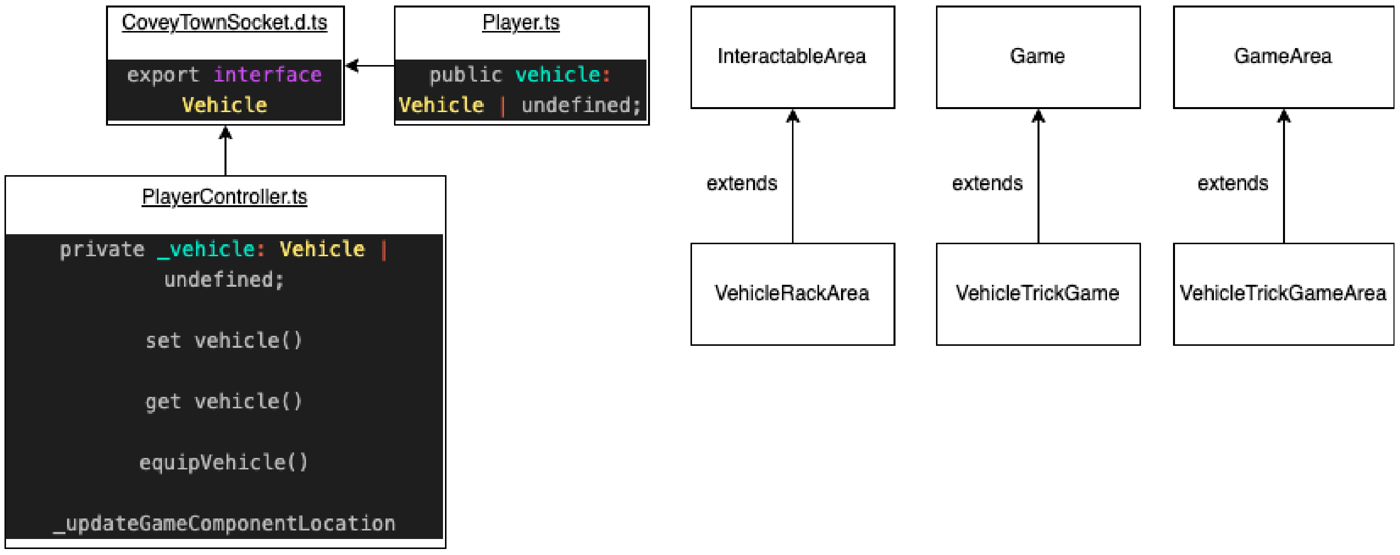
For our vehicle selection rack implementation, our VehicleRackArea extends InteractableArea in the backend, and our VehicleRackArea extends Interactable in the frontend. Both implementations were designed from scratch since the VehicleRack had a new type of Interactable behavior.

For our vehicle mini-game backend, we have a VehicleTrickGameArea that extends GameArea, and the actual game implementation is within VehicleTrickGame. This follows a very similar structure to the way TicTacToe was laid out, which ensures that there is a distinction between the state of our game and the processes that update our game state. On the frontend we have VehicleTrickAreaController which extends GameAreaController, VehicleTrickArea which represents the main window and start page of our game, and VehicleTrick which is the frontend rendition of our game. By utilizing this layout, the code for the game area and the actual game itself is separated and thus easier to debug. The VehicleTrickArea is somewhat similar to TicTacToeArea since it also contains a wrapper and renders the page with the leaderboard and observers, but our implementation also contains a new persistent leaderboard and a fully redesigned UI. The actual VehicleTrick file is obviously very different from the TicTacToeBoard file since they are both separate games.

For our VehicleTrickLeaderboard, we ended up designing this from scratch, although chunks of logic are borrowed from the existing Leaderboard file. The reason we decided to design this from scratch instead of building off of the existing Leaderboard implementation is because even though both leaderboards utilize GameResult values, the leaderboard for our VehicleTrick game has a different GameResult structure with different columns and a different filtering logic as well. We do not have winners or losers, just a player’s 3-letter initials and their score, and as a result our GameResult structure and our leaderboard reflect this. Overall, our vehicle rack and trick minigames utilized good design because they extended existing classes such as InteractableArea and GameArea which didn’t introduce any huge changes to the codebase as a whole.

Here is a diagram to help visualize our code changes:

Backend

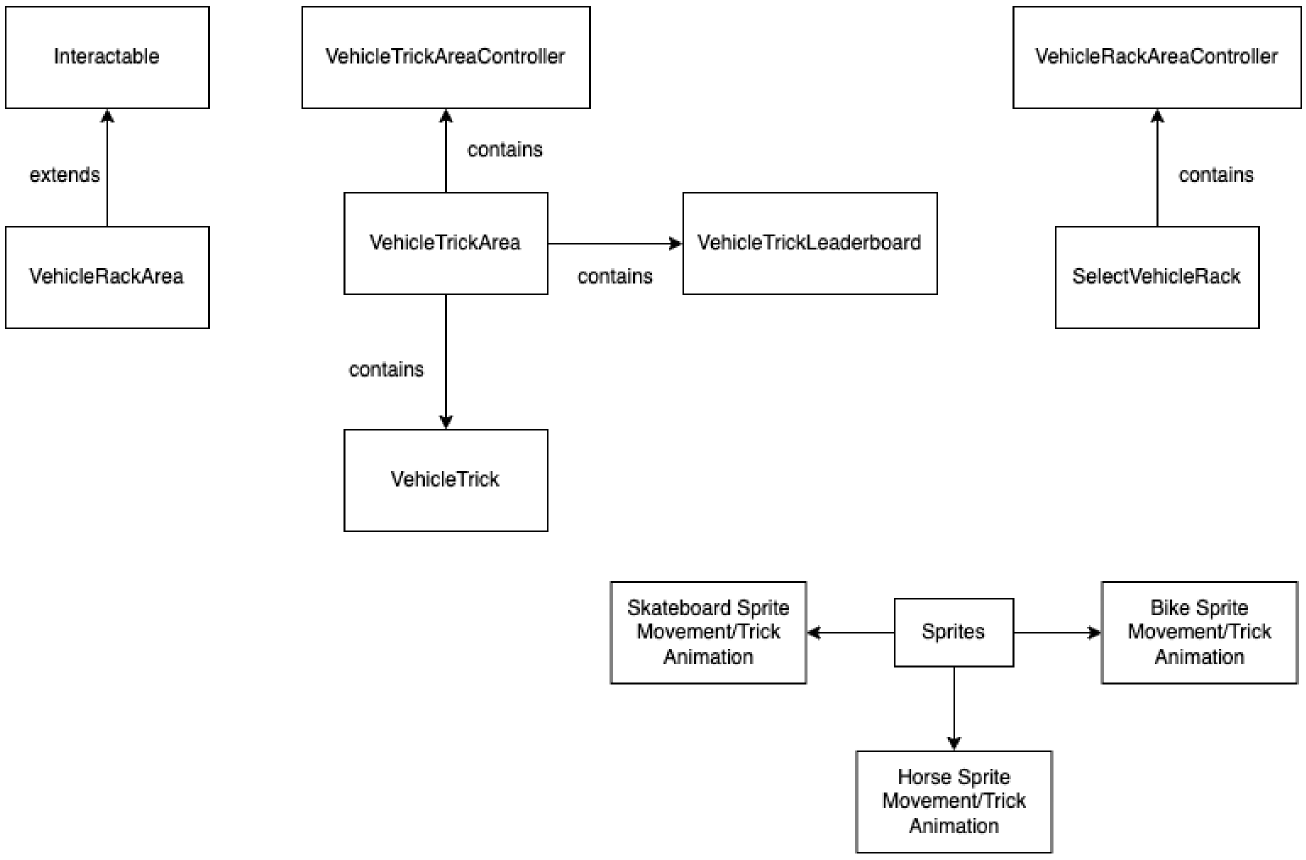


```
export interface
Vehicle
```

```
public vehicle:
Vehicle | undefined;
```

```
private _vehicle: Vehicle |
undefined;

set vehicle()

get vehicle()

equipVehicle()

_updateGameComponentLocation
```

Frontend


Interactable
extends
VehicleRackArea
VehicleTrickAreaController
contains
VehicleTrickArea
contains
VehicleTrickLeaderboard
contains
VehicleTrick
VehicleRackAreaController
contains
SelectVehicleRack
Skateboard Sprite Movement/Trick Animation
Sprites
Bike Sprite Movement/Trick Animation
Horse Sprite Movement/Trick Animation

Process Overview

Our group utilized many agile project management processes. After submitting our revised project plan, we took all of our planned tasks and converted them into Jira tickets on an Atlassian board. From there, we created epics based on our user stories, and divided all of the tickets among the epics. Finally, we assigned the tickets among the group members and split the tickets across 4 sprints (Sprint 0 to Sprint 3). We used the T-shirt sizes for the tasks as point values, and made sure that each member had an equal amount of points for each sprint. We configured the Jira board to display the tickets for one sprint at a time. In each sprint, we had the sections “Todo”, “In Progress”, “In Review”, and “Done” to help us manage our work in the sprint and easily view our progress.

At the start of each sprint, we would have a virtual sprint kickoff meeting to ensure that everyone was aware of the tasks to be completed, and any blockers that we would have on each other. We also met in person every week after class on Thursdays to touch base on our current status in the sprint, as well as any questions or concerns we had about our tickets. We also created an iMessage group chat in which we communicated on an almost daily basis. We used this group chat to ask quick questions, clarify concepts, or to notify other group members that a PR was ready for review. Once we were notified that a PR was ready for review, group members would review the PR with an open mind, and aim to ask questions first before requesting changes. This ensured that PR reviews were blameless and always respectful of the author. We would typically review PRs within a day of them being posted which made the feedback loop very seamless and helped us to complete our tickets faster.

After each of our sprints ended, we would look at our Jira board for the current sprint and see if there were any tickets not in the “Done” column. If there were, we asked ourselves what unexpected circumstances caused this, and how much additional time would need to be spent to complete those tickets in the next sprint. This happened very irregularly for us, but when it did, we made sure that the group members assigned these tickets were given all the help they needed to complete all of their work. Here is a summary of what was planned to happen in each sprint and what actually happened:

Sprint 0

The goal of Sprint 0 was mainly research to see how we could integrate our idea in the codebase. We also planned to define our very basic vehicle data structure which was important to many parts of our feature. All of these tasks were completed.

Sprint 1

For Sprint 1, we planned to create all of the sprites for the vehicles we were bringing into Covey.Town. In addition, we wanted to do all of the backend work for the vehicle rack, equipping vehicles, and adjusting the player's speed based on any vehicles they have equipped. All of the sprite work was completed, but we had two tickets regarding equipping vehicles which were not completely finished yet. This was mainly due to the fact that these two tickets blocked each other in different ways and couldn't be completely resolved especially since Sprint 1 was the shortest sprint. As a result of this, we decided as a team to start our tickets on the first day of each future sprint so that we could find these blockers earlier and resolve them. We also agreed to communicate in the group chat ASAP when one of us was blocked by something.

Sprint 2

Sprint 2 was a big sprint for us since our goal was to implement the sprite movement animations, both the frontend and backend components of the trick game, finish the fronted component of the vehicle rack, and improve our testing. We fulfilled almost all of our goals here, besides creating the sprite for the vehicle rack as it would be seen in the map. This was because the movement animations took us longer than expected since we've never had to deal with them before. However, we overall thought that our planning and execution process this sprint was great as we got a ton of work done. We didn't modify much of our processes for the next sprint except for reassigning a ticket to ensure the animations would get done.

Sprint 3

Since Sprint 3 was our last sprint, our goal was to finish all of the vehicle animations and improve our UI/UX for the vehicle rack and the trick game. After our experiences in our past sprints, we worked very efficiently since we started our work on the first day and communicated with each other very often. Because of this, we were able to finish all of our planned tasks and complete our feature.