Covey Critters
Group 205

Feature Overview:
For this project, we were assigned to implement a new feature into the Covey Town code base. When exploring the town we noticed some similarities in graphics and gameplay to the popular video game, Pokémon. Using this as inspiration, our team decided that Covey Town could benefit by having tiny critters following the user around the game area, like the Pokémon games. After some brainstorming our team came up with three main features we wanted to implement:

1. Carnival Game Area - where the user can play a game to win a critter based on their score
2. Leaderboard - where the user can see how their score compares against other players
3. Covey Critter - a sprite that follows the user with unique characteristics

Our aim was to simulate a game stand that you might see at a carnival, where a player enters a game area then plays the game in order to win a prize.

Running Locally: *https://github.com/wenbakefield/covey-critters*
To run Covey Town locally the user must first download the source code from the GitHub repository and run 'npm install' from the terminal in that folder. Next they can simply run 'npm start' in the /townservice and /frontend folders to spin up a localhost:3000 in their web browser that has our extended version of Covey Town running.

Running from Netlify: *https://coveytown.benwakefield.dev/*
Simply click on the link above.

Entering Covey Town:
Once the user has entered the Covey Town welcome screen, they must submit their username and either create a new town or join an existing town. To create a town, the user can simply submit a town name in the "New Town Name" section and click "Create" to enter a brand new Covey Town. Or, the user can join an existing town by submitting a Town ID in the "Town ID" section or selecting one of the options in the "Select a public town to join" table. Once the user has entered the Covey Town game, they will be greeted with a green "Town *'Your_Town_Name'* is ready to go" Toast notification. Now the user can freely move around the town area using the arrow keys.

Playing the Carnival Game: Add part for first user pick reward rule
Now that the user has successfully entered the Covey Town game, they have the opportunity to play our carnival game and win a Covey Critter. To do this, they must navigate their way over to our Carnival Area which is located at the leftmost wall in the basement labeled Covey Critters. Once the user has positioned themselves in the carnival area, they can click the spacebar button on their keyboard to see a popup modal. If the user is the first person in the town session to want to play this game, they get the opportunity to select the pet rule (the pet assignments based on the score) that the rest of the town will play by. They can select the pet rule by clicking the "select" button, they can then click the "create" button to set the pet rule for the rest of the town. Now any user can walk up to the carnival area and click spacebar to be greeted with a modal that tells the user: the rules of the game, their current rank and highest score (NaN if first time playing), and a "Play Game" button. The user can then click the "Play Game" button to enter a new modal with our spacebar game.

As soon as the user enters the game they will be greeted with a running 100 second timer, a green race track with a sprite at the start line, and a score counter. At this point the user should spam click the spacebar as fast as possible to move the sprite forwards towards the finish line. The game will automatically end either when the timer reaches 0 or if the sprite has reached the finish line before the timer is complete. Once the game is over, the user will see a modal with their final score and a text entry to name their pet. Once they have picked a name they can click the button labeled "Get

Pet!” which will exit the user from the game and award the user a Covey Critter based on the score they got. Along with that, the public leaderboard will also update based on the score the user received.

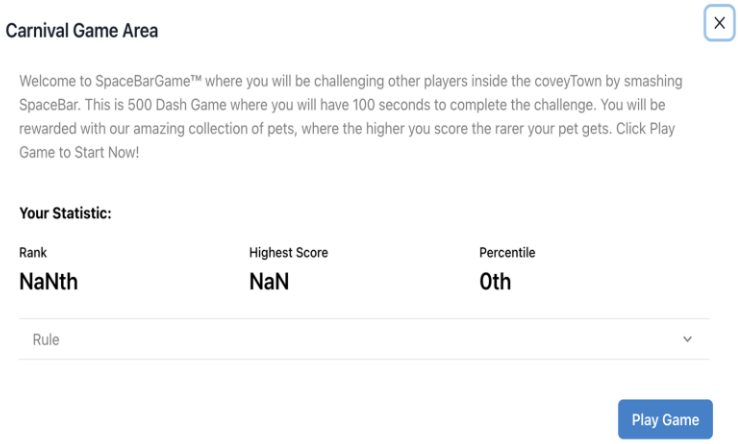


Figure 3: Spacebar game entry point

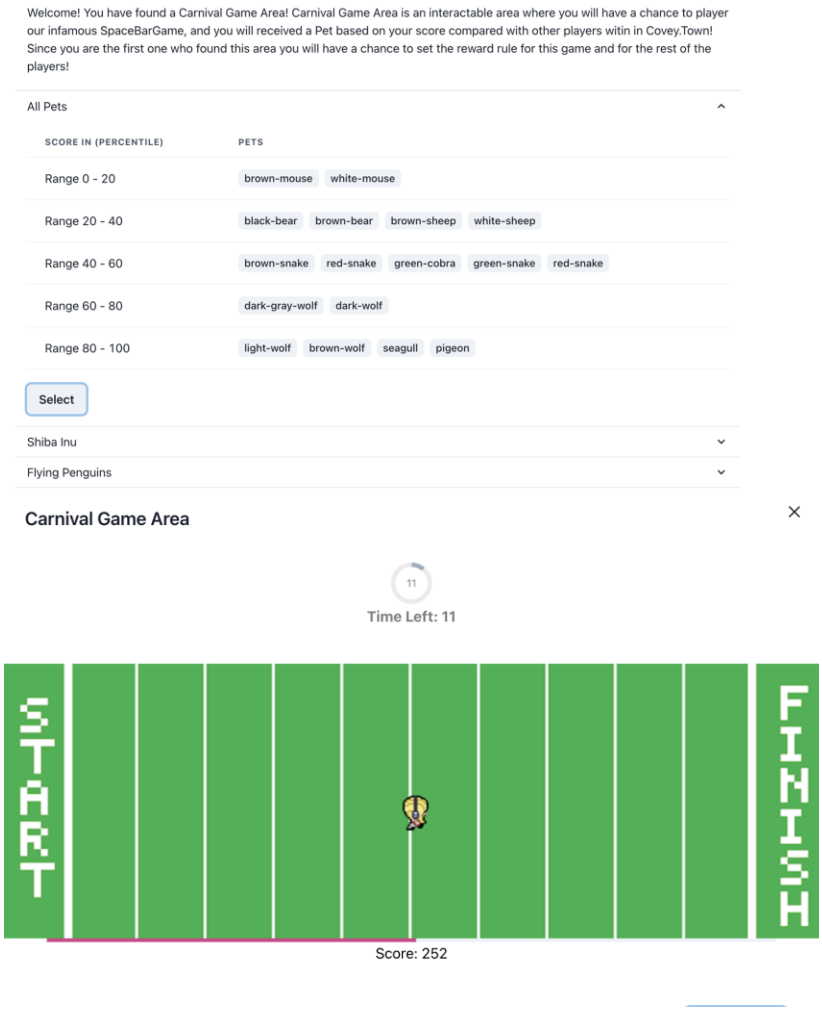


Figure 4: Spacebar Game with timer and score

Figure 5: Covey Critter that follows you around

Figure 6: Game leaderboard

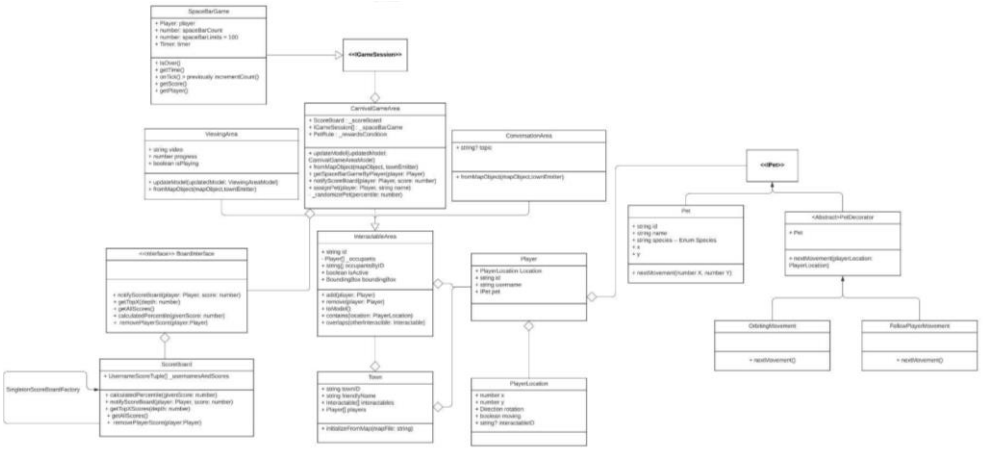
Critter Overview:
The Covey Critters are a collection of 19 different species of animals that can be won after playing the Carnival Game. These include a black bear, brown bear, brown cobra, brown mouse, brown sheep, brown snake, brown wolf, dark gray wolf, dark wolf, gray mouse, gray wolf, green cobra, green snake, light wolf, pigeon, red snake, seagull, white mouse, and white sheep. Each species has a different rarity, so the less interesting animals (such as the mice) are more common than the more interesting animals (such as the birds, which can fly). Each animal's movement pattern is dependent on their species. The birds, snakes, and cobras will orbit around the player as they move, and the rest of the animals will follow behind the player. Each animal has four-frame walking animations for each of the four cardinal directions, except for the birds, which have four-frame flying animations for each direction. After a critter is won, it can be assigned a name by the player, which is displayed over the critter's head.

Technical Overview:
In the backend of Covey.Town we have added 4 main classes to support our 3 main features, including Scoreboard, Pet, SpaceBarGame, and CarnivalGameArea, which will be the area where players will interact with SpaceBarGame.

Design Choice

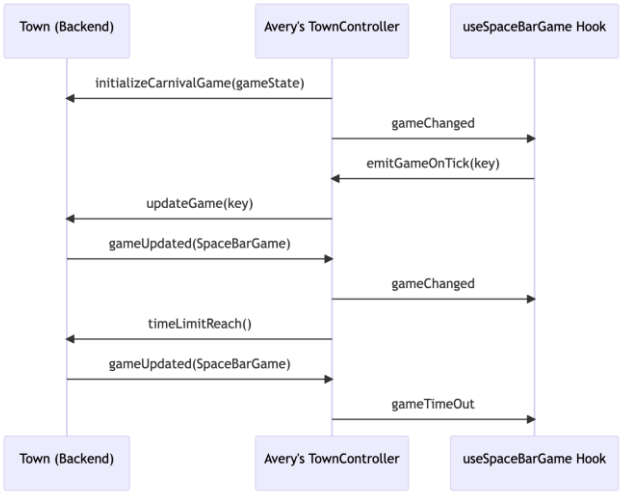
- Singleton pattern was implemented to our scoreboard implementation such that it will only instantiate once, and will be available throughout the different towns in townService. This decision was made due to we want to keep all the active players who played our game in Covey.Town no matter which town they are in Covey.Town
- Decorator patterns such as PetDecorator.ts were implemented in our pet implementation to accommodate various different moving patterns, such as following the player as the player moved through the town, or orbiting around the player as the player moved through the town.
- Factory pattern was implemented to our pet implementation as well alongside our decorator as we encounter that decorator class can be hard to instantiate throughout our codebase, thus we have implemented PetFactory to simplify our codebase

UML Diagram: [Lucid Chart]

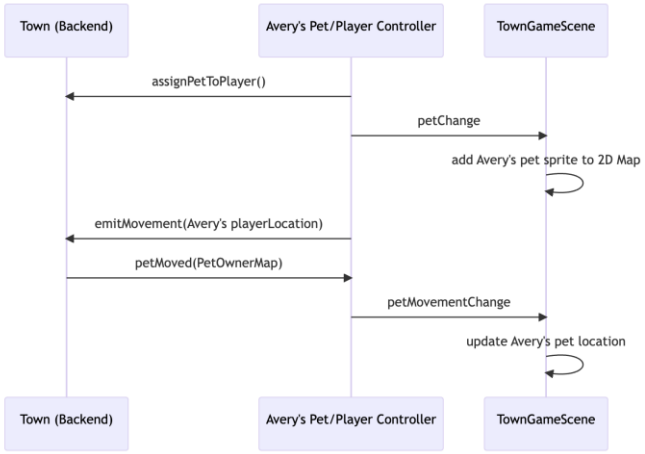


WebSocket and REST API Endpoint

While a user is in a town, communication between frontend and backend for SpaceBarGame and PetMovement occurs primarily over socket API. As we would like a real-time response for both SpaceBarGame and Pet's location when the player interacts with those two components



PetMovement communication between frontend and backend:



Scoreboard communication between frontend and backend primarily uses REST API, where the player Scoreboard re-renders every time the player interacts with the SpaceBarGame. REST API was chosen because the player only needs to see their rank in their leaderboard once they finish the game.

Testing

Automated tests were implemented for the backend features and some frontend features, especially the frontend controller. Although there are some components that cannot be tested with Jest, so Manual Testing is used to test all of the UI components.

Process Overview:

In terms of project management, we conducted multiple iterations of the planning and execution of the project. Since we used agile, most of the plan was divided into multiple medium size sprints, as follows:

Plan Sprints Overview

Sprint	Description
Sprint 0	Designing and Implementing Backend Service
Sprint 1	Implement REST API/Web Socket
Sprint 2	Implement Front End Component
Sprint 3	Fix bugs and work on additional features

Actual Sprints Overview

Sprint	Description
Sprint 0	Design and Planning Phase - Ben Wakefield: Designed Pet Class - Nachiket Gaguly: Designed SpaceBarGame Class - Emre Gucer: Designed ScoreBoard Class - Jirawat Zhou: Designed CarnivalGameArea Interactable Class
Sprint 1	Implement Backend Service - Ben Wakefield: Implement and Test Pet Class - Nachiket Ganguly: Implement and Test SpaceBarGame Class - Emre Gucer: Implement and Test ScoreBoard Class - Jirawat Zhou: Implement and Test CarnivalGameArea, PetDecorator
Sprint 2	Implement REST API/WebSocket and UI Component - Ben Wakefield: Design and Build Pet Sprite and Animation - Nachiket Ganguly: Design and Implement REST API for SpaceBarGame - Emre Gucer: Design and Implement REST API for Scoreboard - Jirawat Zhou: Design and Implement WebSocket and REST API for carnivalGameArea
Sprint 3	Implement Frontend Service - Ben Wakefield: Render Pet onto TownGameScene - Nachiket Ganguly: Implement UI SpaceBarGame Phaser on Frontend - Emre Gucer: Implement UI for Scoreboard and revisited implementation of Scoreboard REST API - Jirawat Zhou: UI Enhancement and Integrating UI Component with WebSocket/REST API

There were some deviations from our Revised Project Plan, where web-socket, and REST API endpoint tasks were underestimated. Therefore some of the tasks such as Integrating between WebSocket/REST API got split between Sprint 2 and 3. In addition to the underestimated WebSocket/REST API tasks, our scoreboard backend and frontend implementation were revised throughout multiple sprints. As a result in Sprint 3, additional resources were assigned to revisit scoreboard implementation.

Sprint Reviews

In terms of the sprint reviews, a meeting was held at the end of each sprint where each developer reviewed each other's code through the pull request that will be merged from their feature branch, denoted with the prefix ‘C[issue number]’ to the development branch. Those reviews will be focused on the technical aspect and tests, where developers who open the pull request will receive feedback on their implementation and decide on any revision according to the feedback. At the end of each sprint, a code-review request is also requested from other developers on the team before merging our development branch with the main (master) branch. In this review, our primary focus will be on integration, and resolving any merge conflict if some code was integrated correctly.

Retrospectives and Blameless Review

Our team’s work pattern had let us achieve the best possible retrospective and blameless reviews. First of all, we spent as much time as possible making the best plan we could. In order to achieve this we discussed every single idea to every detail and compared their advantages and disadvantages to build the plan that suits us the best. As an example, we discussed what type of an area we will build and concluded that a game would be a fun idea, then we discussed what would be our options for games. After both of these discussions, we discussed what type of a reward system we would build.

Our communication through the project helped us to achieve better planning and better blameless reviews. As we communicated with each other the entire time, we could ask opinions of each other about some tasks and it helped us to design the best possible plan for some tasks. Since we collectively designed some tasks, we didn’t have many problems with implementations.

We used git branches to fully extend to have at least 1 branch per task. We used branches to develop and test each small update. Before merging branches with the dev branch we reviewed and tested each others' code. We kept our reviews and comments mostly related to whether the code works or not and whether there is a clear way to improve the code's quality.

Lastly, CI/CD pipeline helped us with the entire testing process. First of all, it allowed us to check whether the merge concluded as we expected it to do so. Secondly, it helped us to test our branches before merging as well and get a preview of the new feature we are adding; therefore, we were able to see results of each branch much more clearly. This is important since this helped us with making better analysis of update; therefore, we could write better comments. The key point of our blameless reviews was writing all comments directly related to the code's result.