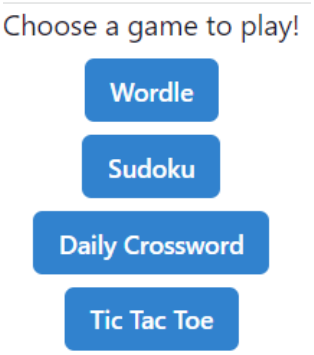
Group 3H Project Report

Abby Cossette, Fen Cullen, Michael Briggs, Mobolaji Rotibi

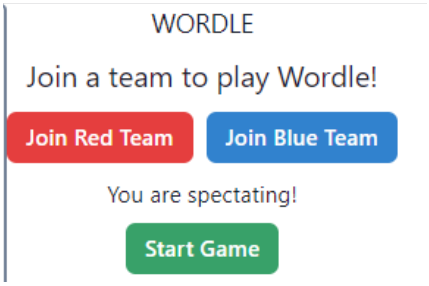
FEATURE OVERVIEW

The new feature we have implemented into Covey.Town is the ability to play games within a conversation area. We also created a team-based Wordle-style game.

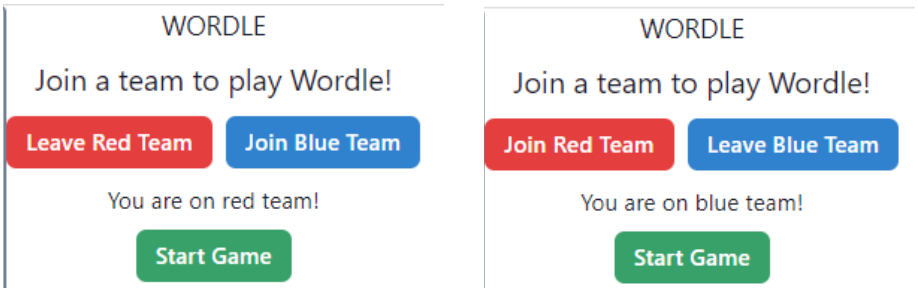
To play Wordle, one must move into a conversation area and set it active by assigning it a topic. A game window then opens on the right side of the existing social sidebar that allows the player(s) to choose to play Wordle.



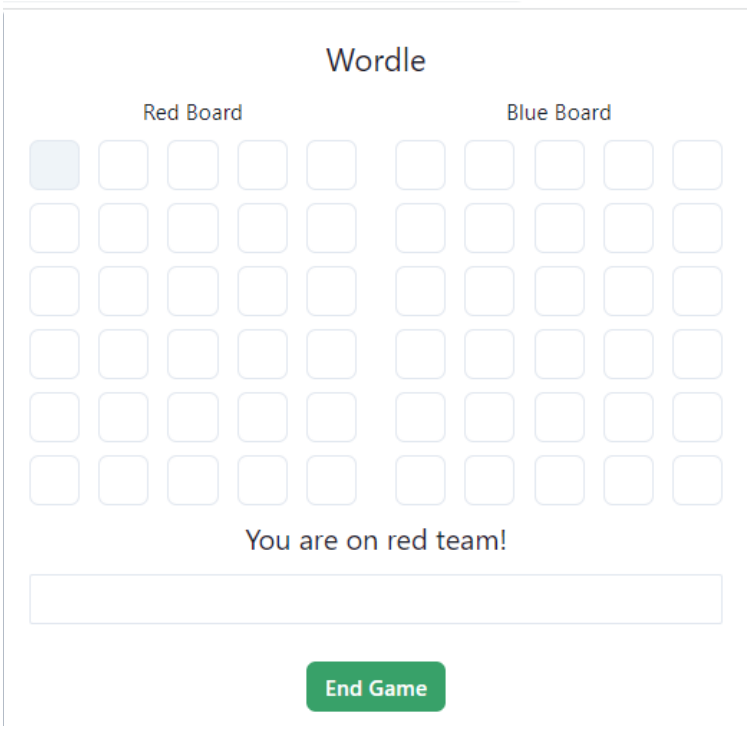
After clicking Wordle, a Game Lobby opens where players can choose to join a team.



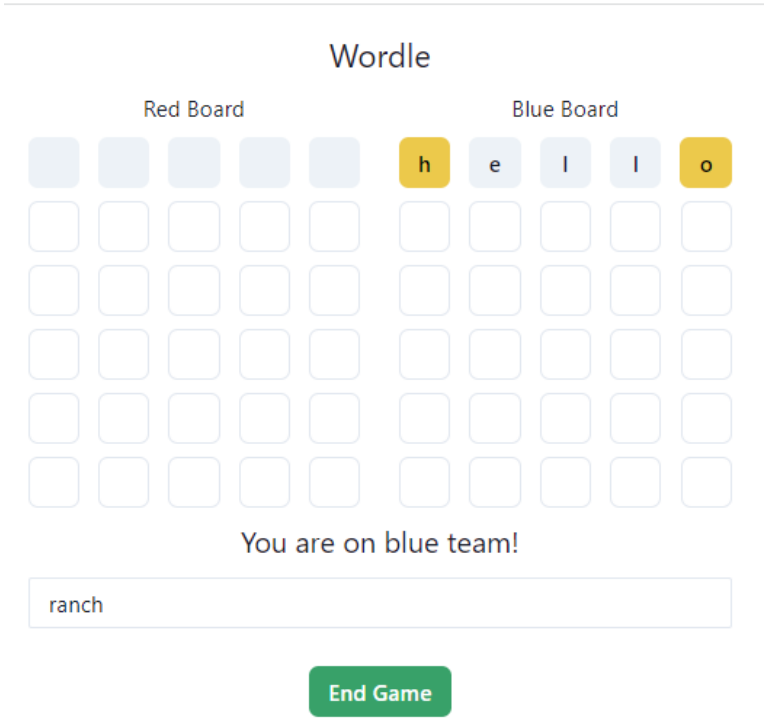
When a player clicks the “join red team” button, they now see the option to leave the red team. The same goes for the blue team.



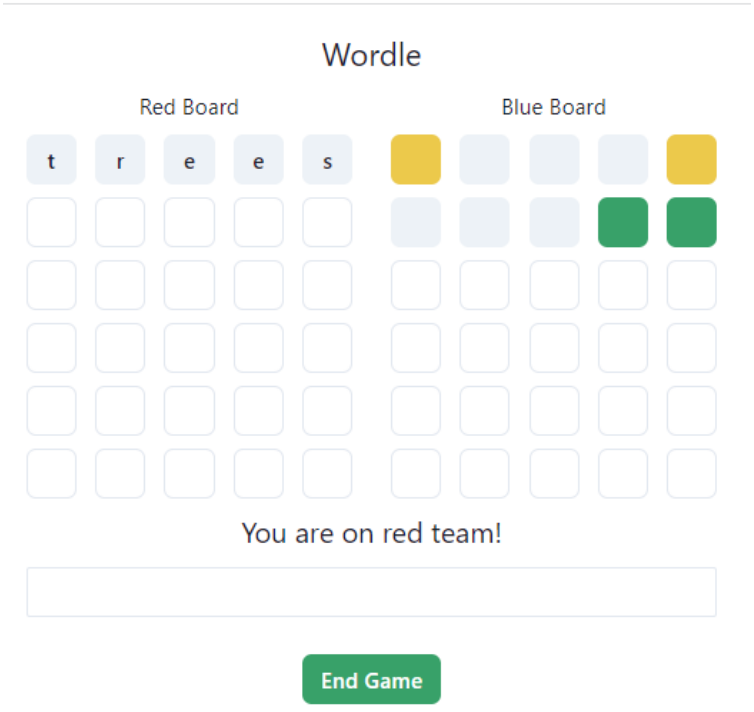
Once teams are finalized, the players can click “Start Game” to launch the Wordle board. Before either team guesses, the boards are empty. Players can enter guesses through the input box as shown below.



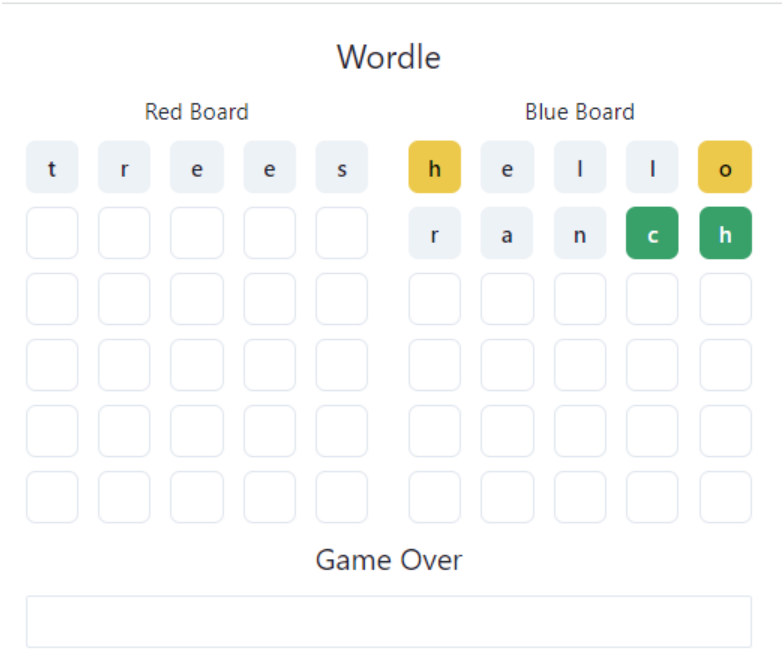
When a player types and enters a 5 letter word, it will register on their board. Players on the blue team can see partial information about the red team’s guesses. They see how many guesses they’ve entered and the color results of those guesses, but not what the actual words are.



Players on the red team also see this partial information on the blue team's board.



At any point, players can choose to end the game by clicking on the “End Game” button, which brings them to the gameover view, where they can see full information about both boards.



TECHNICAL OVERVIEW

We chose to implement our Wordle Game following the classic Model-View-Controller design pattern we learned in OOD, separating most of the game logic from the view. Each existing Conversation Area was awarded game capability with an optional game field of type IGame. Our model is represented by the WordleGame class, which implements IGame. We implemented endpoints to access the state of the game for the frontend to use. Our frontend code consists of a variety of React components, including GameWindow, WordleLobby, and WordleBoard.

CRC CARDS

BACK and FRONT END

IGame	
Responsibilities	Collaborators
Contains logic and state of a game. - setSessionActive	- ConversationArea - WordleGame

- addPlayerToTeam - removePlayer - gameActive - inputAction - getState	

ServerConversationArea	
Responsibilities	Collaborators
Contains additional field - gameModel	- IGame

GameAction	
Responsibilities	Collaborators
Format for a game action, such as a guess input - actionString - playerID - team	- IGame - WordleGame - Various endpoints and request handlers

GameState	
Responsibilities	Collaborators
Represents the state of the game - teamOneState - teamTwoState - winner - isActive / isEnabled	- IGame - WordleGame - Various endpoints and request handlers - Front end Game Board

GameType	
Responsibilities	Collaborators
“wordle” - we made this a type so that our code is open to extension, for example if we later wanted to add a “tic tac toe” game.	- CreateGameRequest

TeamState	
Responsibilities	Collaborators

Represents the state of a team. Includes: - teamMembers - guesses - attemptsLeft?	- WordleGame - Frontend Game Board

Guess	
Responsibilities	Collaborators
Represents a team's guess, with a guessResult array that maps each letter to its display color (gray, yellow, or green).	- WordleGame - WordHandler - Frontend Game Board

BACK END

WordleGame	
Responsibilities	Collaborators
Contains logic and state of a Wordle game. Implements IGame.	- IGame - Player - WordHandler

WordHandler	
Responsibilities	Collaborators
Contains the Wordle dictionary and calculates the result of a guess. - get wordList - get targetWord - handleGuess - randomizer - fillMap	- WordleGame

Existing back end edits

towns	
Responsibilities	Collaborators
Contains additional post routes	

- Create a game - Start a game - Update a game - Get game state - Add player to a team - Remove player from game	

CoveyTownRequestHandlers	
Responsibilities	Collaborators
Contains additional request handlers - gameCreateHandler - gameStartHandler - gameStateHandler - gameInputActionHandler - gameAddPlayerHandler - gameRemovePlayerHandler	

TownsServiceClient	
Responsibilities	Collaborators
Contains additional request and response types - CreateGameRequest - CreateGameResponse - UpdateGameRequest - UpdateGameResponse - GameJoinTeamRequest - GameJoinTeamResponse - GameLeaveTeamRequest - GetGameStateRequest - GetGameStateResponse - StartGameRequest	- GameAction - GameState - GameType

FRONT END

GameWindow	
Responsibilities	Collaborators
Contains a typescript component that displays contents corresponding to the state of the game of the player's active conversation area.	- ConversationArea - WordleBoard - WordleLobby

WordleLobby	
Responsibilities	Collaborators
Contains a typescript component that allows the player to join a team, leave a team, and start the game.	

WordleBoard	
Responsibilities	Collaborators
Contains a typescript component that displays the game board of each team and allows players to enter guesses.	

Existing front end edits

WorldMap	
Responsibilities	Collaborators
Displays GameWindow alongside the world map and social sidebar.	- GameWindow

TownsServiceClient	
Responsibilities	Collaborators
Contains additional request and response types - GameLeaveTeamRequest - GetGameStateRequest - CreateGameRequest - UpdateGameRequest - GameJoinTeamRequest - StartGameRequest Contains additional post routes - createGame - addPlayerToGameTeam - removePlayerFromGameTeam - getGameState - startGame - inputGameAction	- ServerConversationArea - GameType - GameAction - Player

README

Our feature does not require any new setup to deploy. Thus, the original instructions to deploy Covey.Town can be followed, as pasted below.

Running this app locally

Running the application locally entails running both the backend service and a frontend.

Setting up the backend

To run the backend, you will need a Twilio account. Twilio provides new accounts with $15 of credit, which is more than enough to get started. To create an account and configure your local environment:

1. Go to Twilio and create an account. You do not need to provide a credit card to create a trial account.
2. Create an API key and secret (select "API Keys" on the left under "Settings")
3. Create a .env file in the services/townService directory, setting the values as follows:

Config Value	Description
TWILIO_ACCOUNT_SID	Visible on your twilio account dashboard.
TWILIO_API_KEY_SID	The SID of the new API key you created.
TWILIO_API_KEY_SECRET	The secret for the API key you created.
TWILIO_API_AUTH_TOKEN	Visible on your twilio account dashboard.

Starting the backend

Once your backend is configured, you can start it by running npm start in the services/townService directory (the first time you run it, you will also need to run npm install). The backend will automatically restart if you change any of the files in the services/townService/src directory.

Configuring the frontend

Create a .env file in the frontend directory, with the line:
REACT_APP_TOWNS_SERVICE_URL=http://localhost:8081 (if you deploy the towns service to another location, put that location here instead)

Running the frontend

In the frontend directory, run npm start (again, you'll need to run npm install the very first time). After several moments (or minutes, depending on the speed of your machine), a browser will open with the frontend running locally. The frontend will automatically recompile and reload in your browser if you change any files in the frontend/src directory.