Covey.Town: An Integrated Student Project Sequence

Jonathan Bell, Northeastern University

Course website: https://neu-se.github.io/CS4530-Spring-2023/

WMSEE 2023

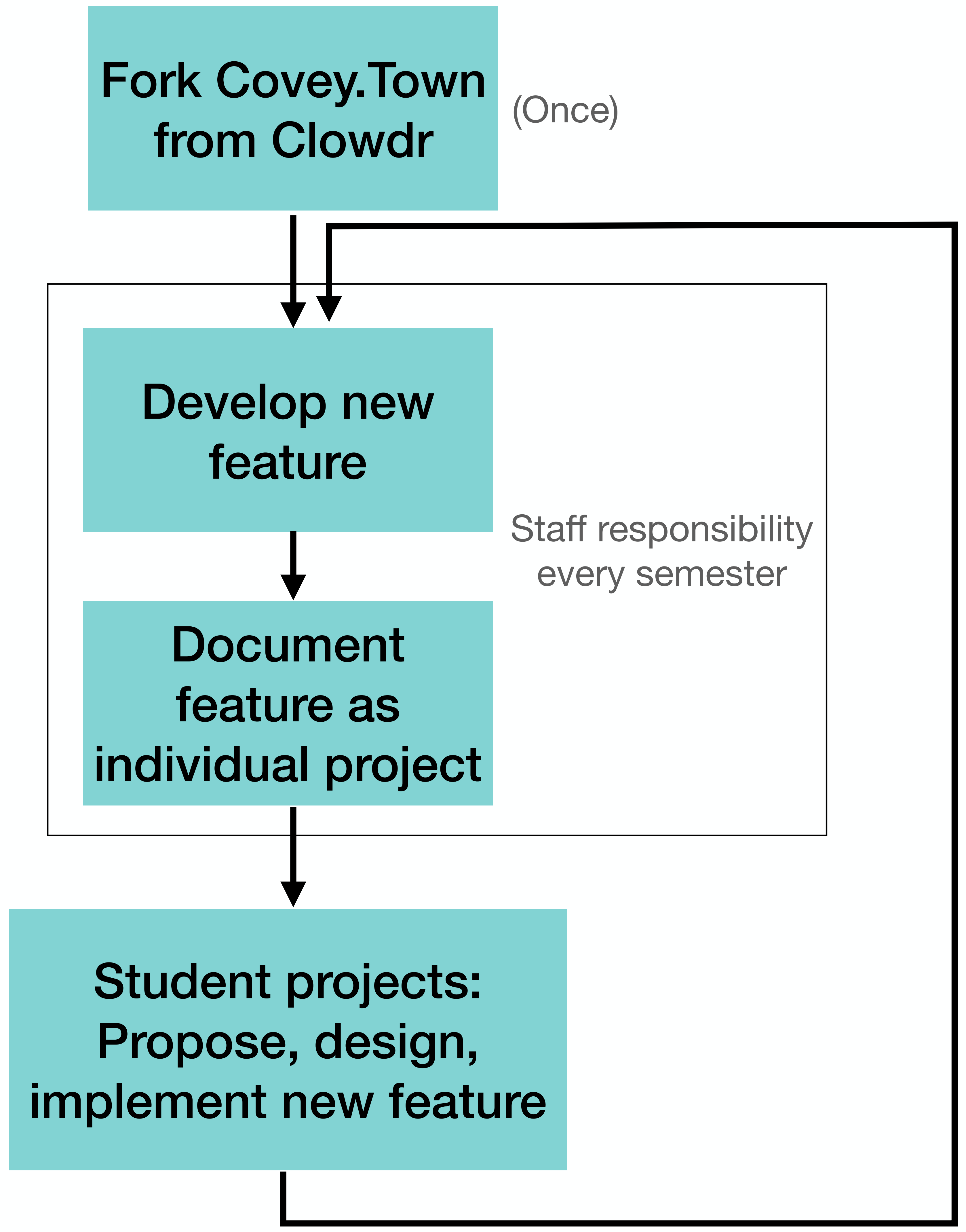
Context: our class (CS4530) in one slide

- Upper-division undergraduate course usually taken in 3rd or 4th year
- Other than object-oriented design, this is it for SE in the curriculum presently
- Intended outcomes:
  - Students will be able to define and describe the phases of the software engineering lifecycle (requirements, design, implementation, testing, deployment, maintenance)
  - Students will be able to explain the role of key processes and technologies in modern software development.
  - Students will be able to productively apply instances of major tools used in elementary SE tasks.
  - Students will design and implement a portfolio-worthy software engineering project in a small team environment that can be publicly showcased to recruiters.

Homework + Project Design Requirements

- Must remain relevant/maintainable over multiple semesters
- Must allow students creative freedom in designing/implementing project
- Must support students with mixed preparations: some familiar with… git, , VSCode, TypeScript, React… others not
- Must *scale*: Spring 2023 had 335 students spread across 7 sections, 3 instructors, 22 TAs
- Resulting structure: Staff-designed individual project (largely auto-graded), then self-proposed group project

Covey.Town Overview

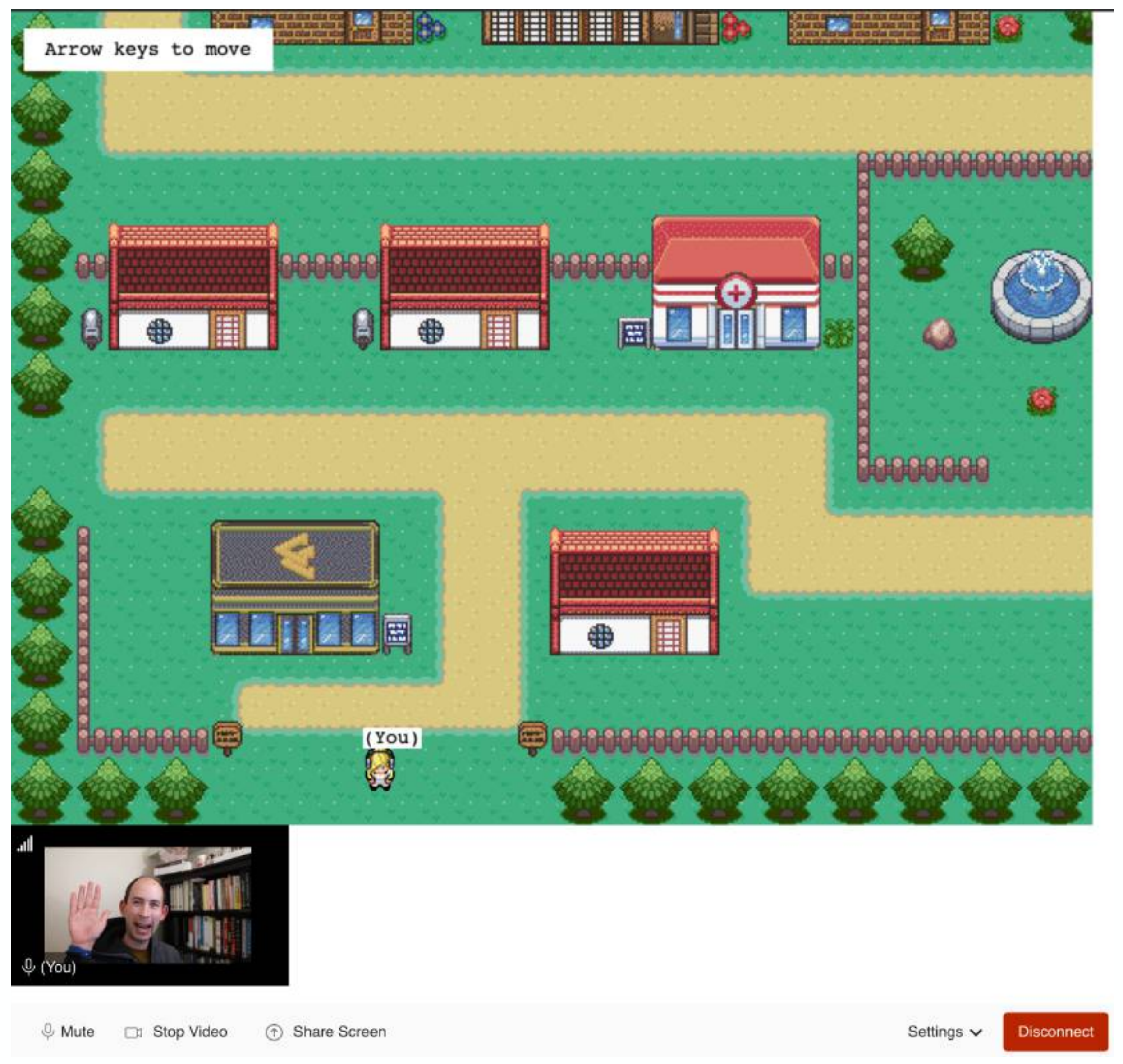


Fork Covey.Town from Clowdr
(Once)
Develop new feature
Document feature as individual project
Staff responsibility every semester
Student projects: Propose, design, implement new feature

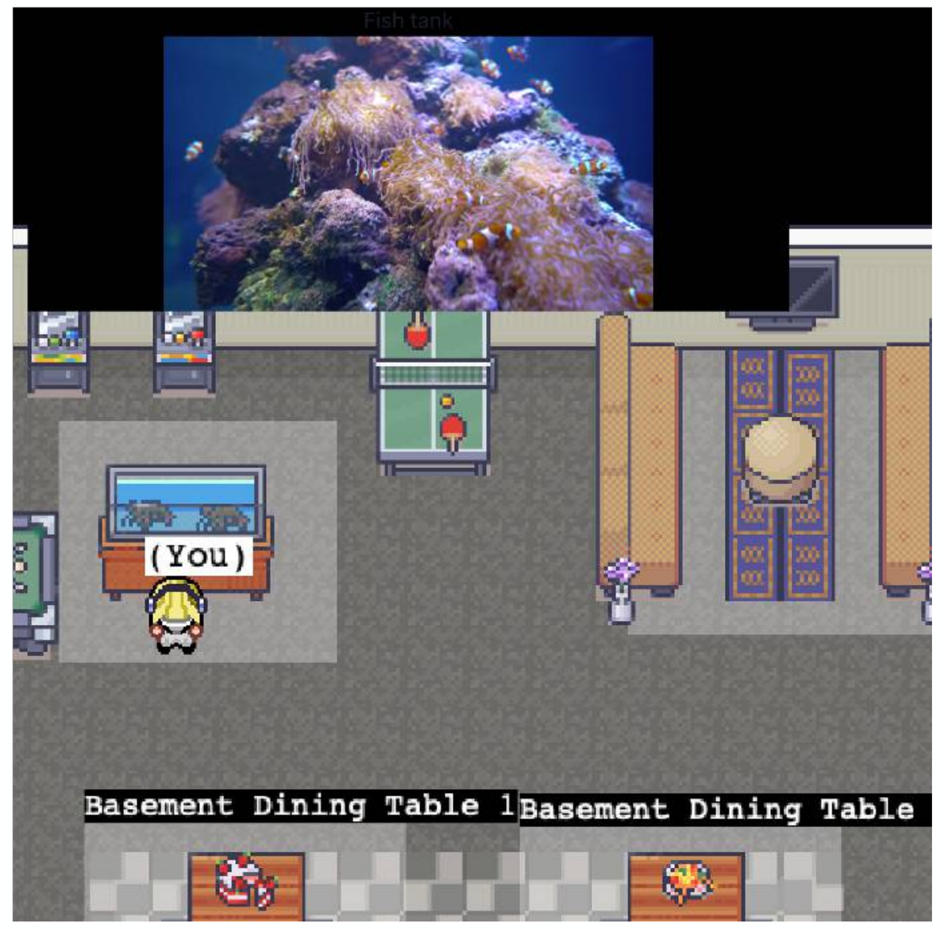
Use student feedback to design new individual project

Accept student contributions as pull requests

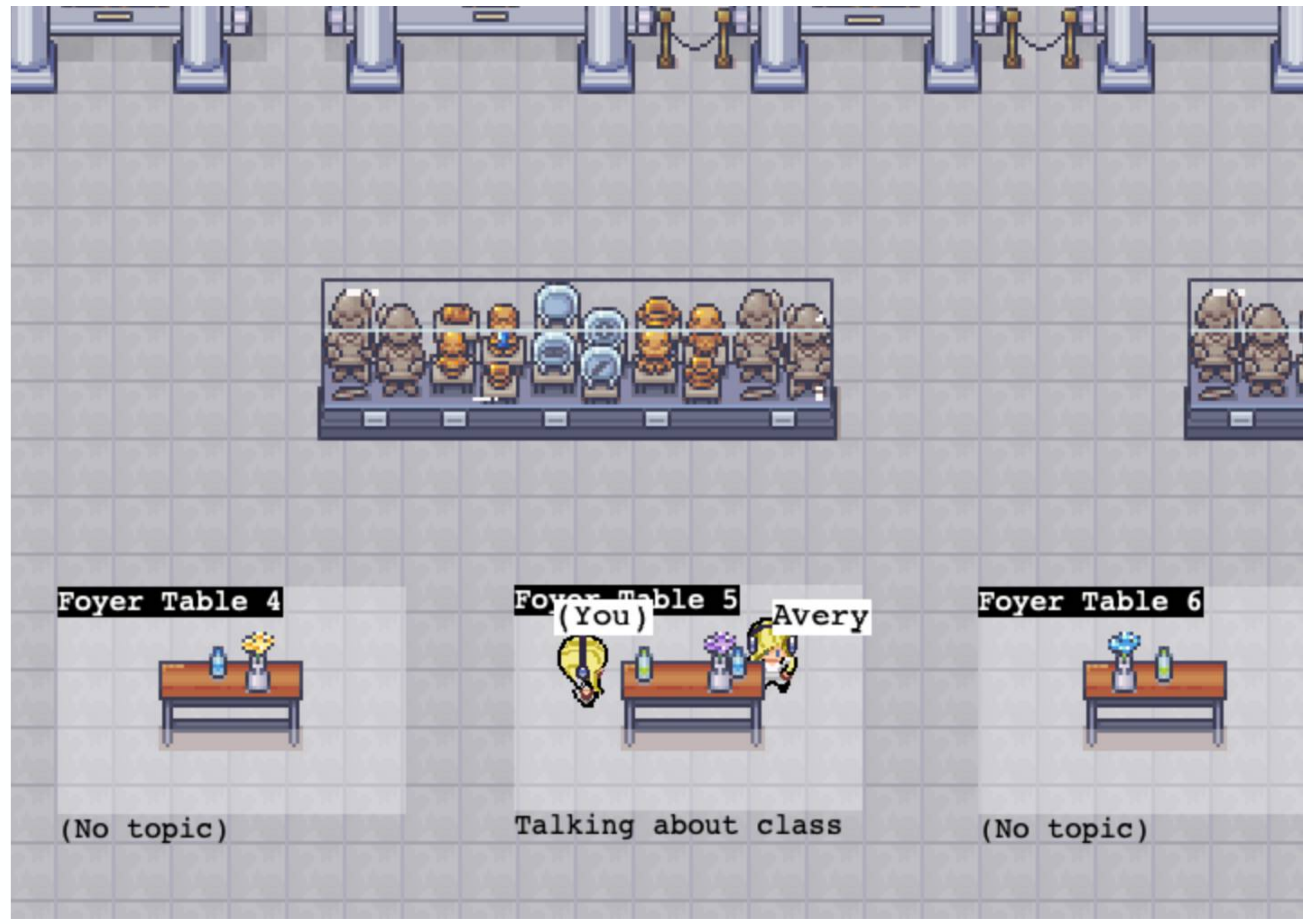
Example student projects:
* Add mini-games
* Text chat features
* Customize the town/world



Spring 2021: Support multiple towns



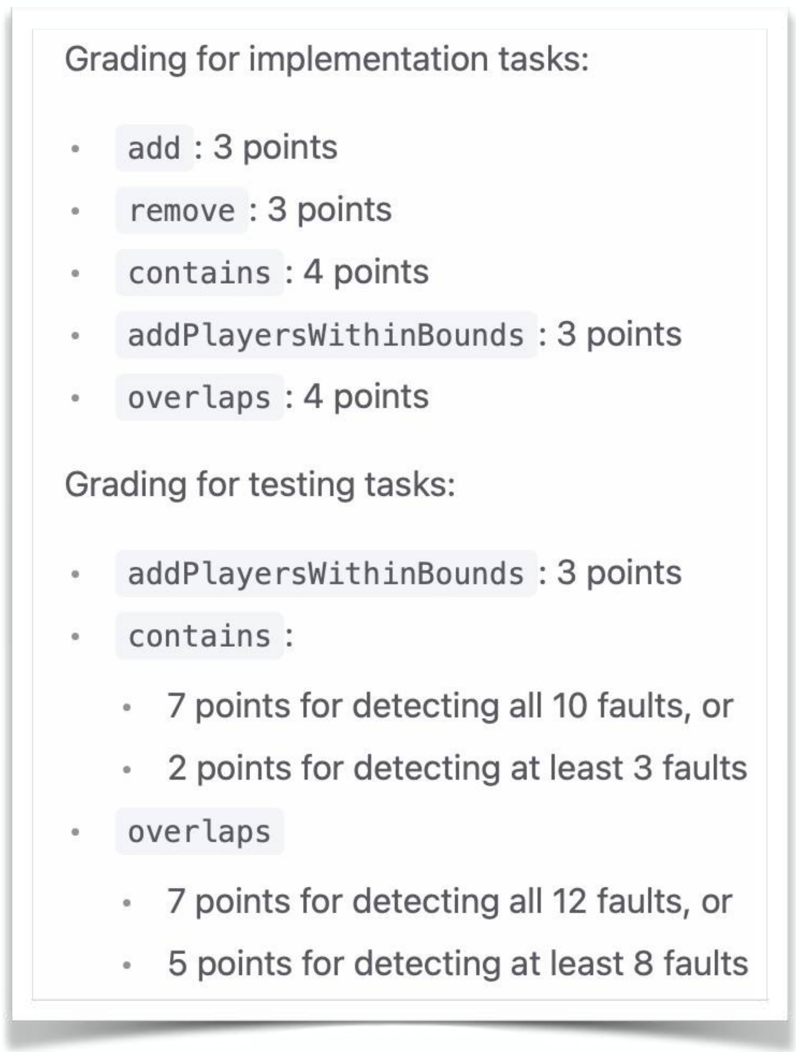
Fall 2022: “Viewing Areas”



Spring 2022: “Conversation Areas”

Individual Projects are Primarily Auto-Graded

- Functional implementation checked by tests
- Test cases checked by mutation analysis
- Code quality checked by linter and manual analysis



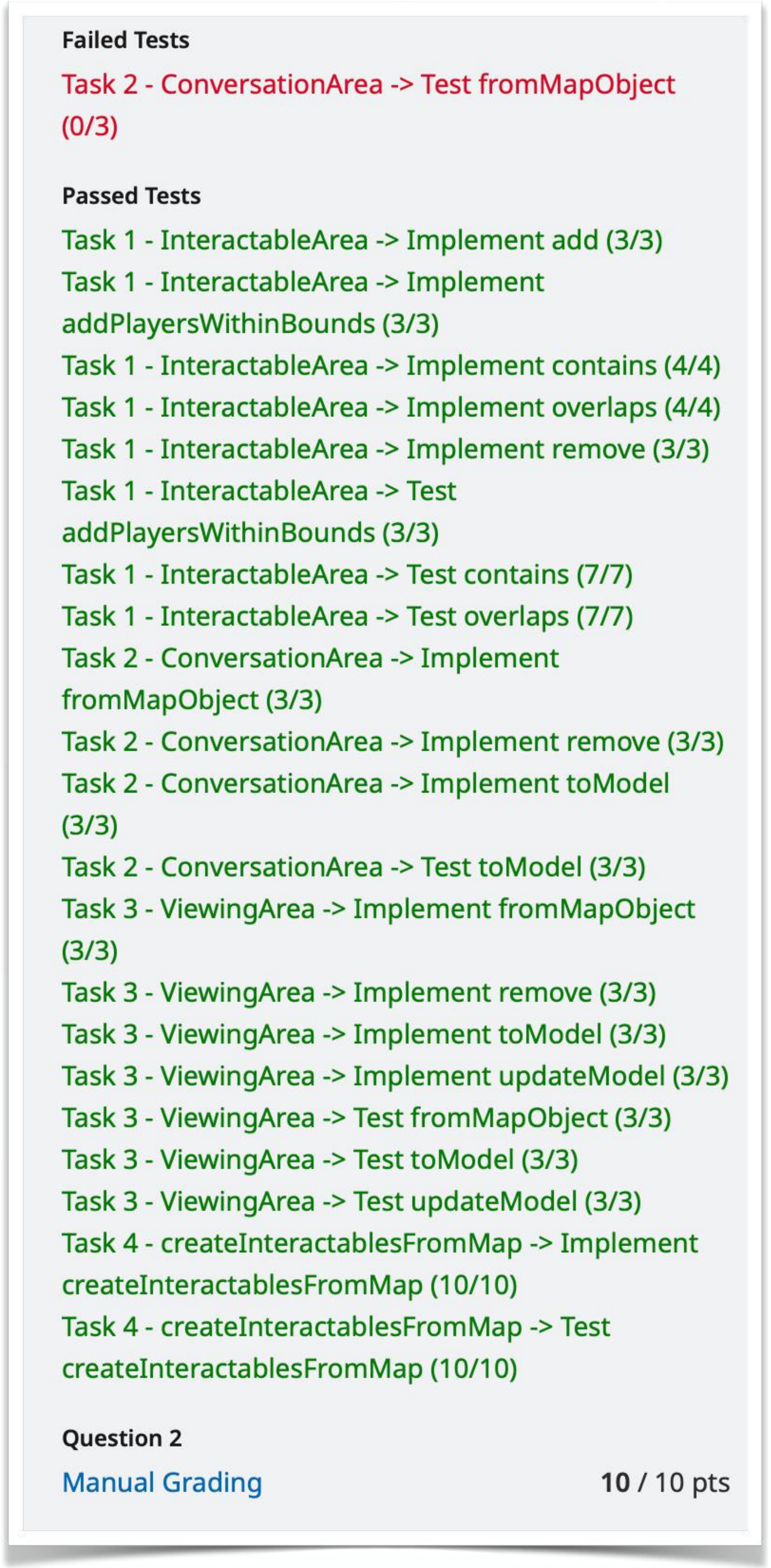
Grading for implementation tasks:

- `add` : 3 points
- `remove` : 3 points
- `contains` : 4 points
- `addPlayersWithinBounds` : 3 points
- `overlaps` : 4 points

Grading for testing tasks:

- `addPlayersWithinBounds` : 3 points
- `contains` :
  - 7 points for detecting all 10 faults, or
  - 2 points for detecting at least 3 faults
- `overlaps`
  - 7 points for detecting all 12 faults, or
  - 5 points for detecting at least 8 faults

Sample rubric for one component



Failed Tests

Task 2 - ConversationArea -> Test fromMapObject (0/3)

Passed Tests

Task 1 - InteractableArea -> Implement add (3/3)
Task 1 - InteractableArea -> Implement addPlayersWithinBounds (3/3)
Task 1 - InteractableArea -> Implement contains (4/4)
Task 1 - InteractableArea -> Implement overlaps (4/4)
Task 1 - InteractableArea -> Implement remove (3/3)
Task 1 - InteractableArea -> Test addPlayersWithinBounds (3/3)
Task 1 - InteractableArea -> Test contains (7/7)
Task 1 - InteractableArea -> Test overlaps (7/7)
Task 2 - ConversationArea -> Implement fromMapObject (3/3)
Task 2 - ConversationArea -> Implement remove (3/3)
Task 2 - ConversationArea -> Implement toModel (3/3)
Task 2 - ConversationArea -> Test toModel (3/3)
Task 3 - ViewingArea -> Implement fromMapObject (3/3)
Task 3 - ViewingArea -> Implement remove (3/3)
Task 3 - ViewingArea -> Implement toModel (3/3)
Task 3 - ViewingArea -> Implement updateModel (3/3)
Task 3 - ViewingArea -> Test fromMapObject (3/3)
Task 3 - ViewingArea -> Test toModel (3/3)
Task 3 - ViewingArea -> Test updateModel (3/3)
Task 4 - createInteractablesFromMap -> Implement createInteractablesFromMap (10/10)
Task 4 - createInteractablesFromMap -> Test createInteractablesFromMap (10/10)

Question 2

Manual Grading **10** / 10 pts

Granularity of grading in GradeScope

Current Status and Open Challenges

- Moving beyond automated grading to reach automated tutoring
- Scaling to multiple campuses (our own Vancouver and Seattle campuses are using this, Martin Kellogg at NJIT has started using this)
- Long-term sustainability of student deployment infrastructure (RIP, Heroku)
- Scaling development of individual project to new TAs
- Reusing old assignments as tutorial/teaching material

Sample Team Projects

CS4530 Final Project: "Tic Tac Toe"

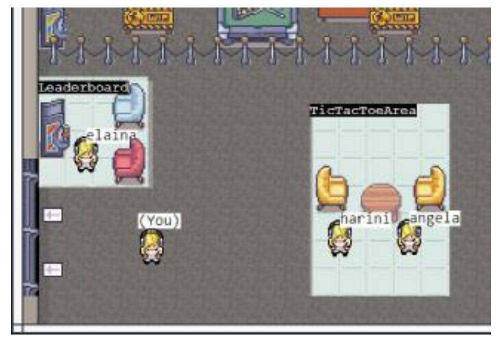
Group 2E: Angela Hu, Elaina Phalen, Harini Boddu, Robin Lu

Our Feature: Tic Tac Toe

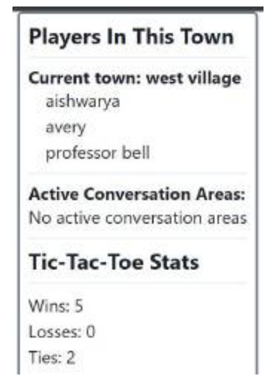
In the original release of Covey.Town, we noticed that users could only have conversations with each other but not interact in any other way. We thought adding a game would be a fun way for players to interact and get to know one another. Our feature implements **Tic Tac Toe** where users can play the classic Tic Tac Toe game with each other. Like conversation areas, users will join a designated Tic Tac Toe area and begin a new game when there are two players. Other users can join the game as spectators. Players can see their Tic Tac Toe statistics and navigate to the leaderboard area on the map to view the leaderboard with top scorers in the town.

Demo & Source

Our demo site is available at https://group-2e-tictactoe.netlify.app and our code at https://github.com/neu-cs4530-s22/team-project-group-2e



The Tic Tac Toe area and leaderboard area are labeled with text boxes. In this screenshot, Harini and Angela are playing Tic Tac Toe against one another, and Elaina is viewing the leaderboard.



Players In This Town

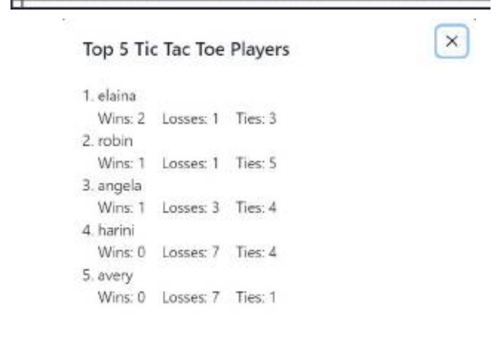
Current town: west village
aishwarya
avery
professor bell

Active Conversation Areas:
No active conversation areas

Tic-Tac-Toe Stats

Wins: 5
Losses: 0
Ties: 2

A player's Tic Tac Toe stats in their Social Sidebar



Top 5 Tic Tac Toe Players

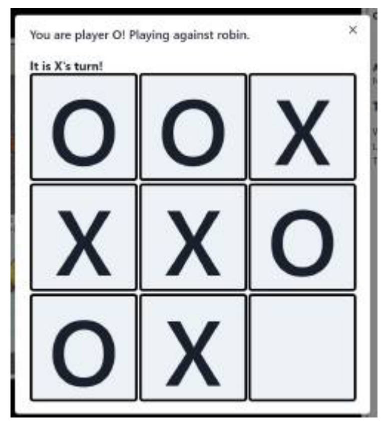
1. elaina
Wins: 2 Losses: 1 Ties: 3
2. robin
Wins: 1 Losses: 1 Ties: 5
3. angela
Wins: 1 Losses: 3 Ties: 4
4. harini
Wins: 0 Losses: 7 Ties: 4
5. avery
Wins: 0 Losses: 7 Ties: 1

The leaderboard modal with the town's top 5 players and their respective stats

Our Technology Stack & Design

We implemented the Tic Tac Toe feature in the existing Covey.Town codebase. There is a Tic Tac Toe area and a leaderboard area, represented as "objects" in the tilemap which can be easily manipulated using "Tiled." These objects are dynamically constructed when the map is loaded and rendered on the screen by Phaser. When a player enters a Tic Tac Toe area and presses space, a React/Chakra modal is displayed inviting them to start or join a game which is input through the modal. When a game starts, a gameplay modal appears and allows the players to take turns by clicking on Buttons. Moves are tracked by the Tic Tac Toe Game backend and synced to each client using socket-io. The player's stats are added to the Social Sidebar which relies on a React hook to receive updates. When a player enters the leaderboard area, a modal appears that rerenders based on a React hook that receives updates about completed Tic Toe Games from the backend.

Our continuous integration pipeline runs an automated test suite in the frontend and backend components and deploys the site using Heroku and Netlify.



You are player O! Playing against robin.

It is X's turn!

O	O	X
X	X	O
O	X	

The current player (O) is playing against Robin (X). It's currently X's turn. Both players have placed moves which are shown in the gameplay modal.

Future Work

Ideally, there would be more than one Tic Tac Toe game going on at once. We would have multiple game areas to play tic tac toe, similar to how there are multiple conversation areas. Currently, our design runs one game at a time, but the town and map store a list of games, so we can have multiple games run simultaneously in the future.

Additionally, we would like to add to the spectator feature to allow spectators to send messages and emotes. These messages and emotes will be visible to the game players and all spectators.

Lastly, we would like to add the ability for players to play again from the "end of game" modal. Currently, they have to exit the area and re-enter which is not efficient if two players want to play a series of games.

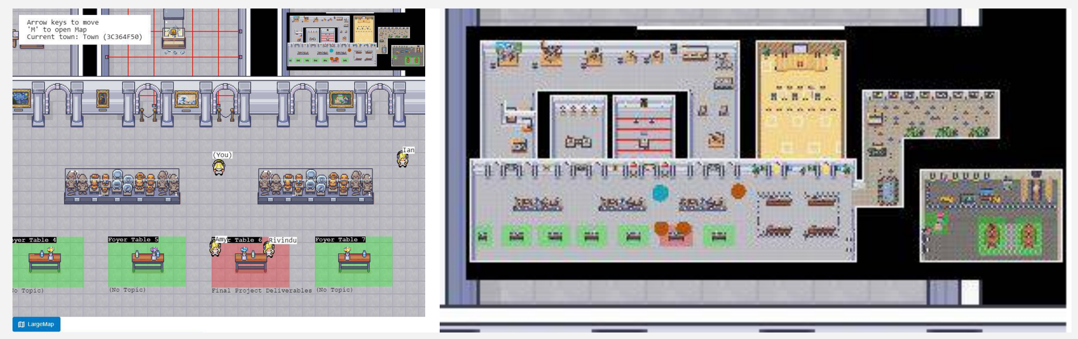
CS4530 Final Project: "Mini-Map"

Group 2H: Rivindu Wijedoru, Amy Min, Ian McLaughlin, Julia Martinez

Mini-Map Feature:

The smaller "mini-map" is always present in the top-right of the normal town perspective, which allows the player to understand where they're located in the town and where they are relative to other users.

Three details were adjusted to make it easier to read and understand: rendering players as dots with the user blue-green and other players orange-brown, filtering text displayed on the town perspective to de-clutter the mini-map, and changing conversation area colors based on their active state.



Left: Full view of the game screen with the mini-map in the top left corner. Foyer 6 is currently active with Amy and Rivindu discussing final project deliverables.
Right: Enlarged view of the mini-map, displaying players as dots.

Design and Implementation:

Three major modifications to the Covey.Town codebase were made. The first consisted of adding a React button component to allow for an immediately noticeable way to toggle between perspectives, or to switch between the normal town and large-map views. This modification consisted of making additional React Context/Provider, state, and hook to help provide access to and keep track of toggle values and updating App.tsx.

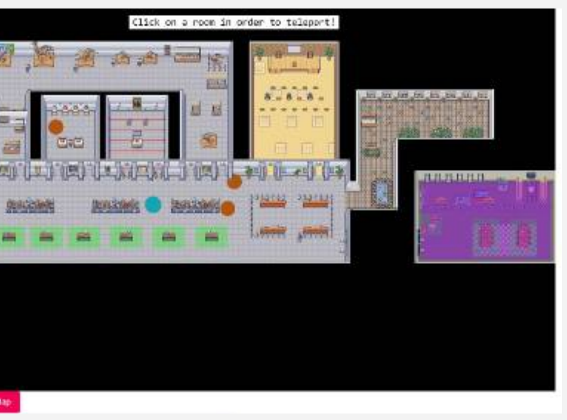
The second consisted of adding two map classes to represent each of the maps we added. Both map classes extended the same Phaser Camera class, but each map had unique dimensions and zoom levels for the maps. The map cameras were then instantiated in WorldMap.tsx's CoveyGameScene, which allowed us to render the maps and ignore and simplify cluttering information from the maps.

The third consisted of adding teleport buttons that would be displayed on the larger map upon mouse hover, which would teleport the user to a set location in the selected room. The buttons were added to the CoveyGameScene, which allowed us to ignore the buttons for all cameras except that of the large map.

Large Map Feature:

The large map was included to provide a bigger view of the town, making additional features, such as teleportation, easier to use. Again, labels and instructions were ignored in this camera view.

The large map can be toggled using the button on the bottom-left or by pressing the "M" key. A user teleports by hovering the cursor over rooms, where a purple button appears for the user to click. The player is then transported into the middle of the chosen room and is brought back into the normal town perspective.



Large map view. The user is currently hovering their cursor over the game room to teleport.

Future Work:

1. To improve our features, we'd *further simplify each of our maps*. The maps would contain only information essential to helping players navigate the town, ignoring further information on the town in the maps. For example, the mini-map would be a solid-fillled shape of the town containing just dynamic components (players, convo. areas) and simplified renderings of town landmarks helpful for distinguishing rooms from each other.
2. To make it easier to modify WorldMap.tsx in future tasks, we would *refactor the CoveyGameScene*, breaking it down into multiple files to shorten the file.
3. Since the maps are intended to help players get a better sense of what's going on in a town, we would *add information labels that would appear upon hover* on the larger map to tell the player, for example, who each player on the map is, and which what people are talking about in active conversation areas.
4. *Abstract the teleport function to any town*. When a player switches to the large-map view, they should be able to select any room in the town and teleport to the clicked location as long as it's an empty tile.

Demo: https://vigorous-rosalind-641d14.netlify.app/
Source: https://github.com/neu-cs4530-s22/team-project-group-2h.git

Why only chat with everyone, when you can chat with some people?

https://github.com/neu-cs4530-s22/team-project-group-2i
https://deploy-preview-42--sad-hamilton-489dda.netlify.app/

Description

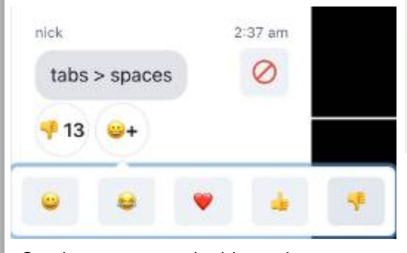
- Adds more chat features to covey.town.
- Send messages exclusively in a conversation area
- Directly message other users in the town.
- React to messages with emojis
- Edit or delete your own messages
- Block annoying users

Design Decisions

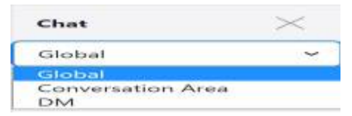
- Separate socket messages sent to add and edit for each type of message
- Editing/Deleting/Reactions all collected into single "edit" socket message
- Blocking players logic handled in frontend (like a personal filter)

Future Work

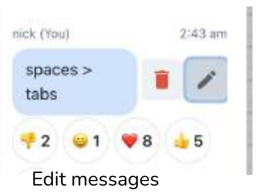
- Seeing all messages that have been sent before you join
- Sending files and images
- Adding custom commands to towns to make chatting easier



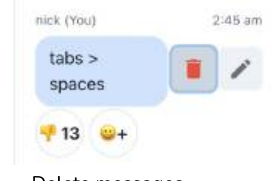
Send messages and add reactions



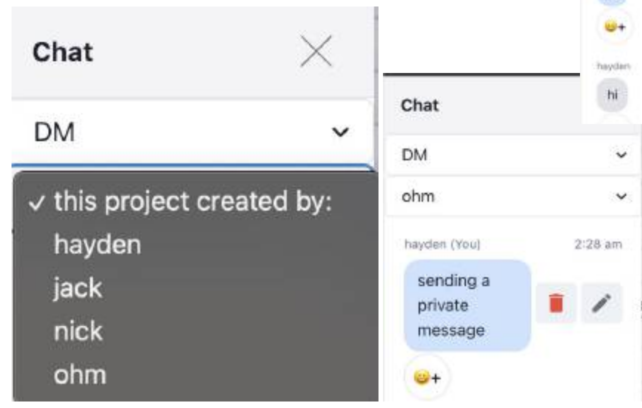
Choose Different Conversation Types



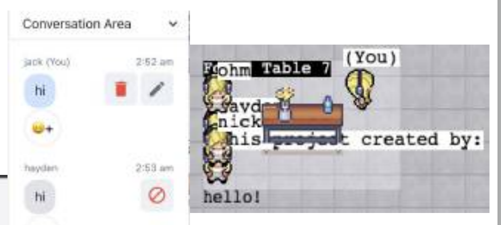
Edit messages



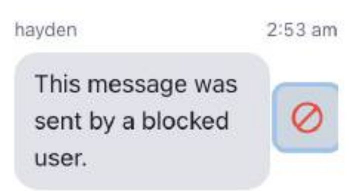
Delete messages



Send Private Messages



Conversation area-specific chats



Block players

CS 4530 Final Project: "Viewing Area"

Group 2M: Aamir Islam, Jonathan Maduro, Jonathan Ju, Petros Papadopoulos

Our Feature: Viewing Area

In a real life communal space, people often congregate not only to chat, but also to watch movies, TV shows, and other videos together for the purpose of entertainment, sharing learning content, or showing off a video created by someone in the space. Since Covey.Town is a place for people to commune virtually and converse, it follows naturally that Covey.Town should support communal viewing parties as another way to interact and share.

It is for these reasons that we developed the Viewing Area feature for our term project. The Viewing Area is a part of the map that users can enter and watch a YouTube video of their choosing together, in synchrony (at the same time). Previously, if a covey.town user wanted to share a video with other users in their server and watch it together, they would have to tell the other viewers to open another tab in their browser and sync up the videos manually, or use another platform which supports synchronous video watching. Now, users can enter a covey.town, walk over to the conference room labeled Viewing Area, type and submit a YouTube video url, and enjoy watching the video synchronously with the other users in the Viewing Area on the same covey.town server. If the video is paused, changed, or fast forwarded or rewinded this change will be reflected in real time to everyone.

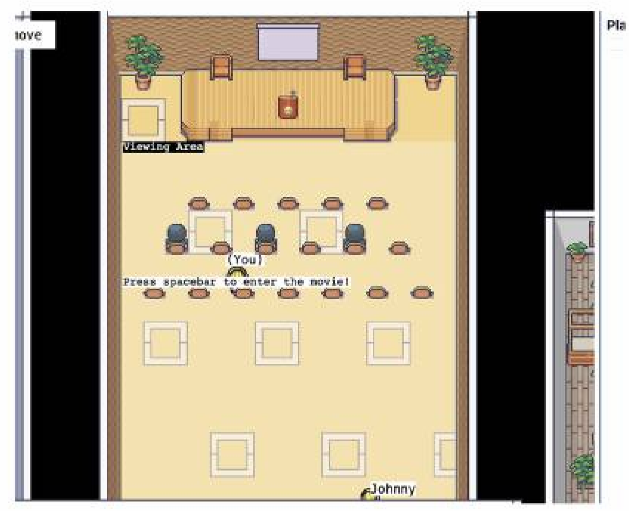
Technology Stack & Design

We implemented the viewing area feature in the existing Covey.Town codebase. There exists a single viewing area represented as an "object" in the tilemap. The object is dynamically constructed when the map is loaded, and rendered on the screen by Phaser.
When a player enters a "new" viewing area, the message "Press spacebar to enter the movie!" is displayed via a Phaser text game object. A React/Chakra modal then appears, with a React/Chakra form component with a submit button embedded within it, inviting the user to enter a YouTube video URL.
Once the user submits a valid URL(our MVP supports YouTube videos only), a video appears with a pause/play button built using a Chakra button component and video progress bar built using a Chakra slider component. The embedded video is built using a ReactPlayer component, a react component which renders a video embed.
These viewing area components are defined in the ViewingArea directory in the frontend, instantiated within WorldMap, and each have hooks linking user interaction to the video status in the backend. The video status is tracked by the CoveyTownController, represented as a VideoStatus type, and synced to each client using socket-io. Any interaction with the viewing area that involves a request to change the status of the video (new url, pause status, elapsed time of the video) will be validated, updated in the backend, and then the change is propagated to every client in the same server. The onVideoStatusUpdated listener is called within updateVideoStatus by the CoveyTownController to propagate any changes made to the video status.
Once a VideoStatus is set in the CoveyTownController, it will automatically increment the video's elapsed time by one second so long as the VideoStatus is not in a paused state and the elapsed time is not equal to the length of the video.

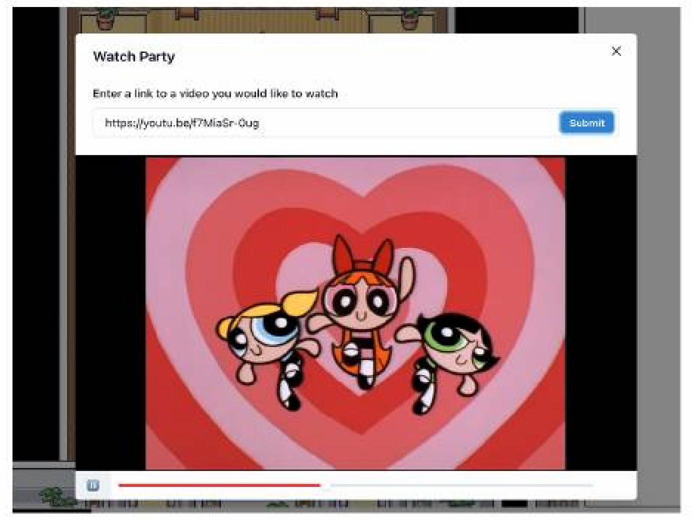
Future Work

When brainstorming different ways to implement a viewing area, we initially envisioned support for any kind of free, publicly accessible video url. To ensure completion of an MVP by the project deadline, we focused on solely supporting YouTube video urls. One decision made along the way was choosing a video embed component that supported only YouTube videos. This among a select few other places was where there was YouTube specific hard-coding in our design. We learned late in the process of a more widely used video embed component that supported any video url, and was easier to work with, and we decided to switch out the YouTube video component with this more general component. Future work could include extending this abstraction to other parts of our design that currently are specific to YouTube, and generalizing them to any video url.

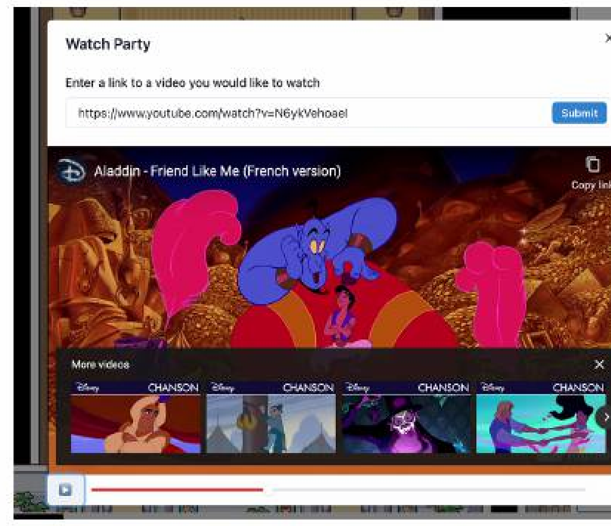
In the future we'd also like to include a full search functionality for videos rather than making a user supply a link to get the video to start playing. This can use the YouTube API, or the API of other video websites, to fetch video suggestions based upon a user defined string. This would allow the user to stay in covey.town fully without ever having to have another window open to actually fetch the links to the videos they want.



Users enter a Viewing Area just as they would a Conversation Area. If they click the space bar, it opens a modal



The modal open and playing a video. The video and timestamp are the same for everyone


Watch Party
Enter a link to a video you would like to watch
Submit
Aladdin - Friend Like Me (French version)
More videos

The modal open with a paused video. It is paused for everyone else in the town too

Try it out: viewing-area.netlify.app **Demo: https://youtu.be/mGJbldblSh0**