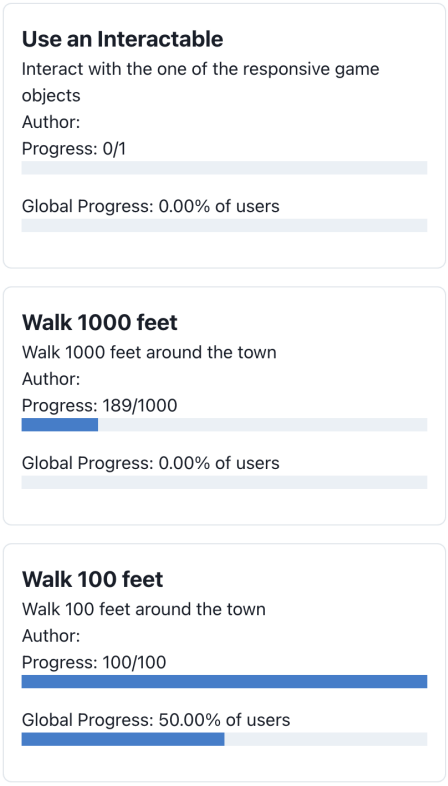
Final Report: Player Achievements!

By George ElMassih, Karim Semaan, and Elisha Mann-Robison.

Feature Overview and User Manual

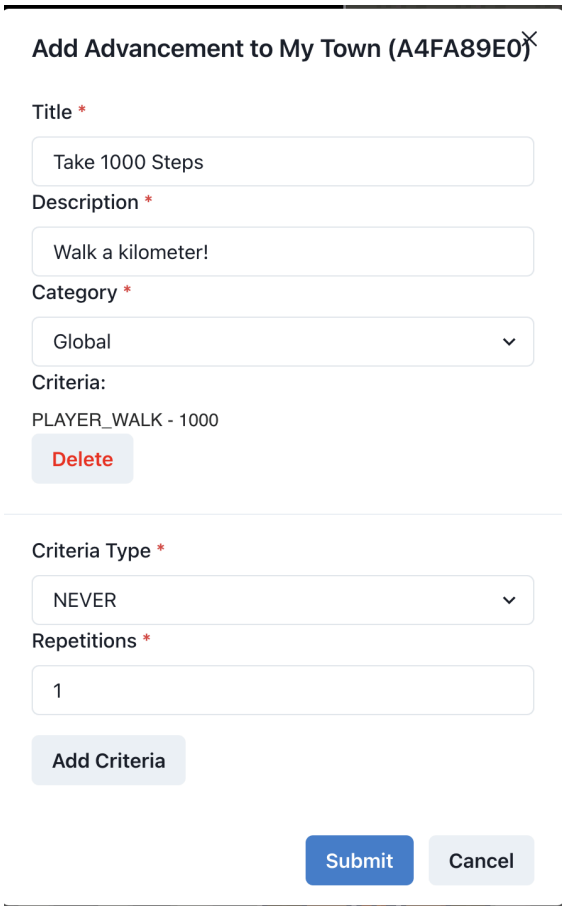
The player achievements feature adds 3 new components for users to interface with - a modal to view current achievement progress, a UI to create custom achievements, and pop ups in the chat when new achievements are earned.

The achievement modal can be opened by pressing “x” anywhere in Covey.town. Within this menu a player can see their current progress for all of their achievements. If an achievement has multiple criteria, the progress bar displays the sum of completion for all criteria. Players may also see the global progress of every achievement - the percentage of users in the town who have completed that achievement. The user may filter this menu in a variety of ways. Achievements from a specific part of the town such as “Tic-Tac-Toe” or “Global” can be filtered. The player can view achievements that have no progress, achievements that have some progress, achievements that have been completed, or all achievements. Players may view another player’s achievement progress as well. The player may also filter by achievements they have created. If the user is an author of an achievement, there is an additional switch to activate or disable the achievement. This switch enables or disables this achievement for all players in the town, along with the user. Achievements are persistent between sessions - even if the user logs out of a town and back in, their achievement progress will remain.



Achievements and progress towards achievements is earned passively while interacting with the world in Covey.town. For example, the user gains progress towards “walking around the town” simply by moving around the world. A player can win interactable games such as Tic-Tac-Toe or Connect Four to accomplish those achievements. When an achievement is completed, a popup appears in the town chat informing the user of their accomplishment.

Players can design original achievements to share with other players. From the menu at the bottom of the screen, a player can select “Add Custom Achievement”. The player can write a title and description for their custom achievement, along with the category of achievement. The player can define as many criteria as they would like for their custom achievement. Each criteria consists of the type of task that must be accomplished, the amount of times this task must be accomplished to finish the achievement. These custom achievements are made available to all players within this town.



Developers can also add achievements across every town by manually adding new achievements and criteria as JSON files. To create a new type of criteria, the developer would add to the “AchievementCriteriaType” enumeration in the backend “CoveyTownSocket”. In the “TownService” file in the backend, developers can trigger their new criteria when appropriate. For example the criteria could be triggered when a certain game is won, or the player accomplishes some task. To contribute progress towards an achievement, that event should be accompanied by a call to achievementManager.trigger(EVENT_TYPE_YOU_CREATED, playerId).

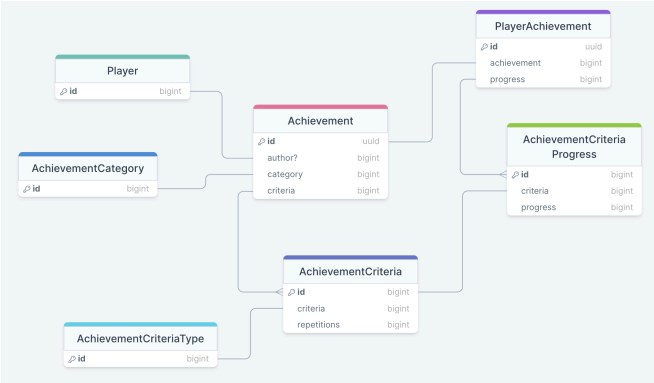
The live deployment of the Achievements feature is available at https://spring24-project-team-416-wa5u.onrender.com/
Players can immediately use all of the features of the achievement system. Simply interact with the world of Covey.town to accomplish progress towards achievements, and press ‘x’ to view the achievements modal. View the chat as achievements are completed to see the pop up, and navigate to the bottom of the screen to add custom achievements.

To build a working version of the project locally, download the source code from https://github.com/neu-cs4530/spring24-project-team-416. Set up the necessary environment files with Twilio certification as described in the README.txt instructions. Then, launch a terminal and navigate to the “townService” folder. Run “npm start” to run the backend. In a separate terminal, navigate to the “frontend” folder and again run “npm start”. The application will be launched and can be accessed from a web browser at localhost:3000.

Technical Overview

This project had major changes in all three zones of the project: the backend (townService), the frontend (frontend), and the API (also townService). We chose to make the townService responsible for listening, tracking, and reporting changes in a player's achievements. This is the right choice because the townService already handles every event that the client has, validating and notifying the other clients. Listeners for our criteria sit in the middle of that logic, triggering the moment the event is validated on the server side. We chose this over client-side tracking because it is not only simpler, but also more reliable from an anti-cheat or even ping-synchronicity perspective (client side logic can result in unsynchronized data between players).

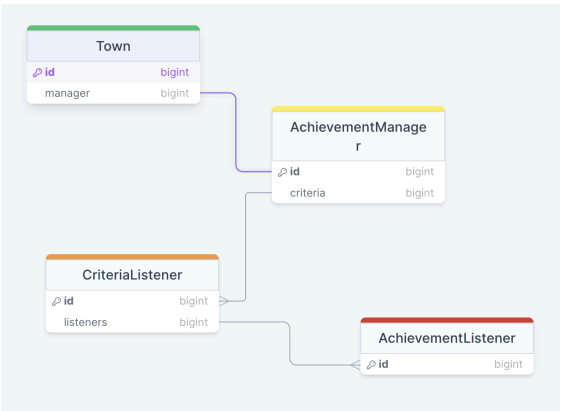
We added several data types to the shared CoveyTownSocket.d.ts file. Specifically, we added an AchievementCriteriaType, AchievementCategory, AchievementCriteria, Achievement, PlayerAchievement, AchievementCriteriaProgress. See diagram:



The central feature is the Achievement data type. An Achievement has a category, author, and a list of AchievementCriteria, which is an extensible type and a number of repetitions. These repetitions allow for more variety in achievements.
Players have PlayerAchievements, which have an Achievement and progress of that achievement (see diagram for specifics).
This data system is structured for extensibility, and is therefore very abstract. This makes the system a little complicated, but allows more flexibility in future extensions.

The server interacts with this data with a waterfall of listeners. The AchievementManager - created and managed by the Town - is responsible for loading the achievements from JSON files when a town is created, handling criteria triggers, updating player achievement progress, and reporting when players get achievements to that player's client.

When the Town (or another part of the townService) notices a criteria has been triggered (it gets a "playerMoved" event, so it triggers a "PLAYER_WALKED" criteria event), it calls the AchievementManager's 'trigger' method. This then begins a cascade of listener triggers, see graphic.



The AchievementManager will find the associated CriteriaListener for that AchievementCriteriaType, and trigger it. It, in turn, will have a list of AchievementListeners which are listening to it (these are the Achievements that have this AchievementCriteriaType in their criteria lists), and trigger those. These AchievementListeners are registered by the AchievementManager on

spin-up. The AchievementListeners look at the data of the player who triggered the event, and up-tick the player's progress appropriately.

The particular challenge with this system was allowing for all of this to occur without overburdening the structure with event handling. If we checked every game tick to see if any criteria had been triggered for any achievement, we would be making potentially thousands of (extremely redundant) checks every tick. Instead, we opted for a single listener per AchievementCriteriaType, capping the number of top-level listeners at a very low amount. We then implemented AchievementListeners, so that when a CriteriaListener triggers, it can instantly alert all Achievements that care about it. Rather than sift through a potentially large list of achievements, we use the AchievementListeners to pre-sort the data at spin-up. All of these changes allow for many Achievements to exist without any server lag occurring.

The next major technical change was to the API. Three main API endpoints were created: (1) GET '{townID}/achievements', which lists all achievements available in the given Town, returning a list of Achievement objects. (2) GET '{townID}/achievements/player/{playerID}', which gets the player's progress on all achievements available in this town, returning a list of PlayerAchievement objects, but with the Achievement data stripped back to just the IDs. The PlayerID is optional, and if left blank will default to using the achievements associated with the session token. (3) GET '{townID}/achievements/global/{achievementID}' returns the percentage of players in this town who have completed the given achievement.

We chose to strip the Achievement data down to an ID in the '/achievements/player' endpoint to reduce the amount of data being sent. This packet will be requested quite often, since opening the modal will trigger a new request for the most updated data, so trimming it down. Further, this is more efficient than having the server notify clients of updates, since those will happen very very often.

The specific additions to the frontend were an AchievementModal, to display the achievements of a given player and a notification system (a toast to notify the player when they acquire a new achievement). More frontend was added in the custom achievements section, but that will again be discussed later on. The above frontend components are very clearly partitioned by function, and handle the things they need to with ease. We think these choices are fairly self explanatory, and no remotely viable alternatives were present.

The final major set of changes was for custom achievements. Four new API endpoints were created: (1) GET '{townID}/achievements/categories', to fetch the available AchievementCategories, (2) GET '{townID}/achievements/criteria', to fetch the available list of AchievementCriteria, (3) POST '{townID}/achievements/create', to register a new achievement, (4) POST '{townID}/achievements/{achievementID}', to "activate" or "deactivate" a custom achievement you created (display or hide it in the list of achievements for this town). On the front end this comes with the AddAchievementModal, a form for creating custom achievements. Since submitting this adjusts the list of achievements the frontend caches, we made the frontend re-query for that list if it detects a progress element associated with an ID it doesn't recognize.

Proposed Feature	**User Stories**	**Status**
Gain achievements	US 1	Complete
See completed achievements	US 1	Complete
See progress on incomplete achievements	US 1	Complete
See unstarted achievements	US 1	Complete
See progress in completing all tasks left to try	US 1	Complete
Notification when a new achievement is gotten	US 1	Complete
Earn global achievements	US 1	Complete
See achievements another user has gotten	US 2, US 3	Complete
See leaderboard of players that have gotten most achievements	US 2	Unstarted
When looking at their achievements, see their leaderboard placement	US 2	Unstarted
See global percentage of players that have completed a given achievement	US 3	Complete
Add custom achievements	US 3	Complete
Open and close achievements for a given amount of time	US 3	Complete
Use in-game UI to create custom achievements	US 3	Complete

Process Overview

Since our project is essentially a large vertical slice to add to the overall project, we attempted to break it up horizontally, to allow individual team members to “own” a horizontal slice of the added piece. Frontend for George ElMassih, API for Karim Semaan, and server/townService for Elisha Mann-Robison. Some aspects of the project necessarily violated these divisions (custom achievements, for example, required full-stack construction, but were completed in whole by Elisha). To keep track of tasks (and to avoid duplicate work being done) we leveraged a SCRUM board - a Github Project linked to our repository, constructed by Elisha.

We used sprints, as recommended by the project design, and leveraged time-boxing within those sprints to try completion of new features or to decide how much to assign to a given person for an upcoming sprint. During the process we leveraged blameless reviews as well, performing (and logging) sprint retros where we maintained complete blamelessness while examining what would be carried over into the next sprint.

In our first sprint, we planned to focus on infrastructural setup - we met to construct an agreed upon data type, set of API points, and goal for the sprint. We then each attempted to construct that infrastructure for our respective horizontal components. Good progress was made in some areas, but for several locations, much of the infrastructure remained incomplete and was carried over into the second sprint. The second sprint had been planned to hone in on implementing and connecting our horizontal components - implementing bells and whistles features, writing extensive integration tests, and working together to perfect the infrastructure inter-communication. Since some of our infrastructure work was carried over, API and frontend sections were bogged down during this sprint by catch-up work. Ultimately, infrastructure work was in a good place by the end of this sprint, but more work was needed for the linkages. At this point, we had yet to do any manual end-to-end tests, which made us nervous.

In sprint 3, we had intended to be honing in on features like custom achievements, fanciness in the UI, and leaderboard data calculations. Since we had a lot of necessary features left to do, we couldn’t do all of those as easily. Nearing the end of the sprint, we were very busy working every day on custom achievements, basic modal UI (filtering, activate/deactivate custom achievements), and (very non-negligibly) the numerous final presentation/poster/writeup/reflection/survey that are simultaneously due at the end of the last sprint. This lagged us once again, but we put in as many hours as we could to get all of our features in.

As can be seen, our sprint retros were vital to our process. We re-envisioned our future sprints every time a sprint ended, and it allowed us to stay on top of our work, without piling an impossible amount of work onto our developers. We strongly recommend this pattern - horizontal stacking (to allow specializing for developers), sprints, retros, a SCRUM board. We found these processes to be extremely useful for organization, motivation, execution, and planning as we worked through this massive project.