CS5610 Final Project Documentation v.1.0: Rogue Mushroom Hunter Game [link]

Dang, Ruohan
dang.ru@northeastern.edu
GitHub: ruohandang

Kuang, Yuan
kuang.yua@northeastern.edu
GitHub: yuankuang

Owens, Jasmine
owens.ja@northeastern.edu
GitHub: jnkowens

Zhang, Yue
zhang.yue20@northeastern.edu
GitHub: willyz

Updated: December 15, 2022

Contents

1 Executive Summary

This project supports a browser-based javascript game. The software consists of a simple login system, a game page, and a leader board.

The game is a mushroom-themed rouge-like, where the player controls a character on a map. The map spawns monsters to attack the player, and the player must destroy the monsters while also avoiding them in order to level up.

To play, users must first register and login to the site. Then, users may access the main game page, where they can start a new game and view the top 5 posted scores.

2 Background

This project was chosen due to a common interest in learning how to code a javascript game. We chose a rogue-like game because the mechanics of this game-type are easier to implement while still being fun to play.

We hoped to fully implement a playable rouge-like game with customization options. Specifically, we wanted user to be able to customize their in-game avatar and choose a map wallpaper. We also wanted to implement common game mechanics such as a level-up system and pickup of objects with special effects. Due to technical challenges with other parts of the software, the game portion has been scaled back to a basic playable demo with no character customization or leveling.

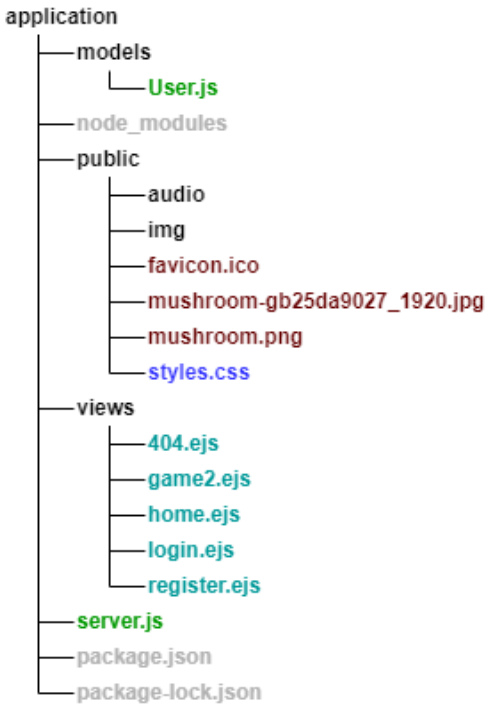

Figure 1: Filetree

The centerpiece of the project is the game script. The rest of the project consists of structures to support the game page. We built a home page and login system that requires users to register a username in order to access the game page. We also did the page design and styling from scratch.

3 Development

The project is built using the Express framework with a MongoDB database. The project architecture can be described using the Model-View-Controller (MVC) pattern, with the web pages representing the view, the Users database representing the model, and the server representing the controller. The project can also be roughly divided into a front end—the web pages—and a back end—the server and the database, though this division is less precise than the MVC pattern.

The web system is meant to be accessible by anyone anywhere for the purpose of playing the game, though it should be noted that mobile devices are not well supported currently.

3.1 Back end — database

For the database, we used MongoDB cloud version – Atlas. This allows us to access databse information from any machine, which was convenient for build-testing.

The database schema has just three attributes for each entry: username, password, and score. The username and password are String datatypes, and are not sanitized in anyway. For a proper release, we would want to limit exactly what characters could be input to these fields.

The database schema is stored in the `Users.js` file in the `models` folder. The database itself is accessed via a link in the server file. Passwords are stored securely in the database. Sample datum:

```
_id: ObjectID("alphabet soup goes here")
name: "username"
password: "alphabet soup goes here"
score: #
__v: 0
```

We considered storing game files (avatar, monster sprites, etc.) in a separate database, but ultimately just kept these assets in local files since we want users to run the game fast and smoothly. The scope of this game is small enough that careful organization of files is not really required. It is unclear that placing these in a database would have a particular benefit in any case.

3.2 Back end — server

All server code is contained in the `server.js` file. The server runs on port 3000.

The server processes user data from input forms to send to the database and handles http requests. It also contains an api function that lets the game page send a highscore to the database.

Early in the development process, the sever was built to render ejs views through Express. We were able to successfully integrate React with the server, which removed the need to render pages in Express. However, the React rollback forced us to return to the ejs view engine.

3.3 Front end — web pages

The project supports four pages: Home, Register, Login, and Game. We also created a custom 404 layout for pages that do not exist in the sever domain.

The page layouts live in the `views` folder. User navigation between pages is

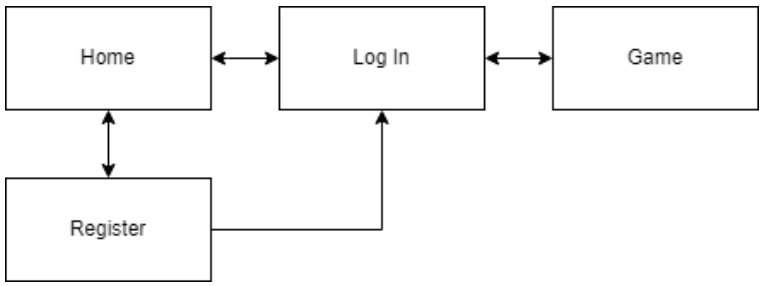

Figure 2: Web Page Navigation

managed through links on the pages, with the exception of user registration and authentication redirects (these are handled by the server). The styles sheet, site images, and game sprites are stored in the `public` folder, and are added to each webpage by Express when the page is rendered.

We began by creating demo pages in html and css, then attempted to port the pages to the React framework. Though we were able to set up the page routing and layouts, overall this effort failed; it proved to be too difficult to embed the game script in React in the time we had left to combine the pieces of the project. When we rolled the development project back to Express-only, we then had to rebuild the pages in ejs.

3.4 Front end — game script

The game script ended up being the biggest stumbling block to the development plan. The script is based on the structure of the game we did in class/as homework, and is all contained in a single script tag.

We attempted to embed the script in React in numerous different ways, and failed each time; it seems that React is really not intended to host custom scripts, as every recommended method Google could produce ended in failure. We could create a canvas element, even from a different file, and embed that in the webpage, but drawing on the canvas using the script proved to be impossible. Likely a solution would require rebuilding the entire script natively in React, which we did not want to attempt this late in the development process.

4 Game design

4.1 Industry background

This game falls in the Bullet Hell, Rogue-like, and Pixel Graphic categories, the combination of which is a recent trend in newly released games.

This type of game typically lets the player control a shooter, which is often but not strictly vertically oriented. The focus is on eliminating enemies while dodging enemies and their projectiles. To make this feasible, the player is usually provided

Figure 3: Example game: Vampire Survivors

a tiny hitbox, highly visible bullets, and various useful mechanics for managing the enemy's fire.

The player will be able to level up and choose new abilities/items/companions by killing enemies or exploring the map. These add-ons change the gameplay significantly when the player focuses their build in a specific direction. With the addition of a good storyline and various maps, this type of game can have a great replay value for players as they try different combinations of add-ons to beat the highest score.

4.2 Game features

Our game uses mushrooms as the main theme because all of us love hunting and eating mushrooms! Images are sourced from free image asset libraries; specifically, from Itch.io | Tyler Warren Favorites, Itch.io | Witch Kitty, Icons8.com, and Pixabay.com.

Current gameplay features

- The player can use arrow keys or w-a-s-d to control the survivor.
- The survivor shoots bullets at the enemy with the earliest spawn time.
- Monsters are generated at random points at the edges of the screen.
- Monsters chase the shooter.
- The player has 3 lives. Each time they touch a monster, they will lose 1 life.

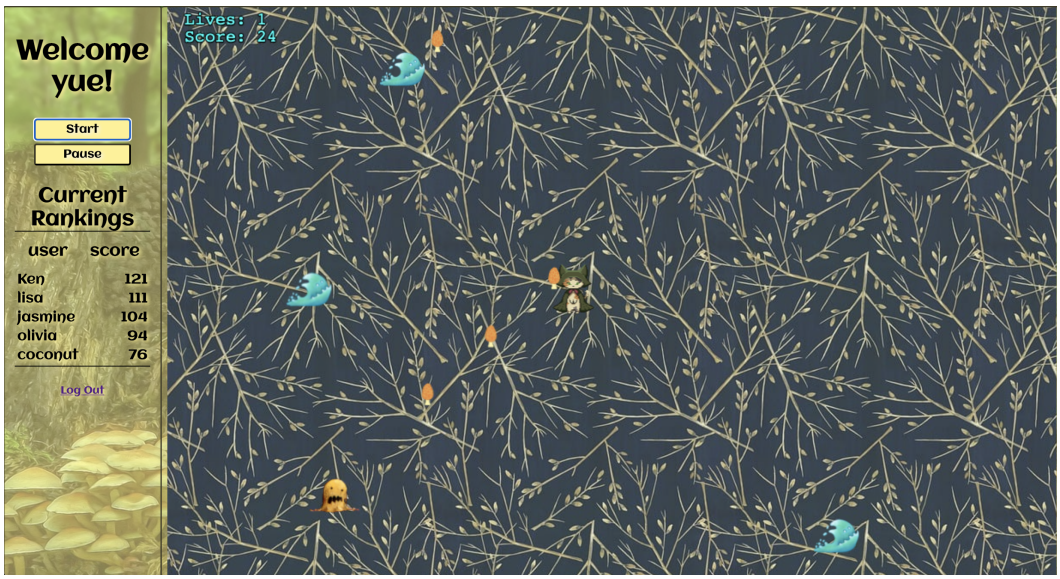

Figure 4: Rogue Mushroom Hunter Game

The game is over when the player runs out of lives.

- The score is how many monsters have been killed before the game is over .
- The highest user score will be stored for the leaderboard.

Planned gameplay features

- The survivor shoots at the closest enemy instead of the earliest.
- Monsters can shoot projectiles.
- The survivor can level up by killing the monsters to gain new abilities (eg: survivor walks faster, bullet moves faster, bullet kills more enemies).
- Multiplayer mode
- More animations
- More music
- Customizable maps

5 Completeness

Throughout the project, we realized that we needed to do less and less for the demo in order to meet the critical targets. We met only one secondary goal, and even had to drop some core goals. The following core goals from the original design document were dropped:

Users

- User avatar selection and pet selection, no customization
  - One choice of avatar
  - One choice of pet

Game

- View Window
  - Centered on user character
  - Pans around larger game map
- User Character
  - User gains EXP on enemy death
  - User upgrades (damage only)
- Enemies
  - Enemy damage calculation
  - Enemy scaling (spawn rate, HP)

The project is a little rough to be considered a full release, but it does demonstrate the basic required functionality. Now that the basic features are working adding more enhancements should be simpler.

6 Reconsiderations

As we have mentioned earlier, the decision to try to use the React framework with a .js game script was the root cause of most of the development problems. Were we to redo this project, we would want to research better ways to host a game in a webpage—perhaps there is a better way to do it with a language other than javascript, or perhaps there is a game-specific framework that would be easier to integrate with a game script than React was. It is also possible that there is a game-specific framework that integrates with React that we were unable to locate during development.

Apart from this one major issue, we are quite pleased with the results of the project, and are enjoying running up the scores on the leaderboard so that anyone who tests the game functionality during grading will have a hard time posting a new high score.

The project is currently hosted on a live server at:
https://mushroom-game-app.onrender.com/home