Team Earthquake

Nobel Gautam, Farhan Haque, Gabriel Marks, Elias Milborn

Overview

We are creating an online multiplayer top-down shooter. After you register and login, if you are the first one online, you are placed in a waiting queue until another player comes online. When another player comes online, you are placed into a game. The game consists of movement with WASD, and clicking to shoot. When you damage your opponent down to 0 health by shooting them, you get a point. Whoever reaches 5 points first wins the game. Your stats for the game and your win are recorded under your username.

Components

File Structure

```
earthquake/
    app.py
    templates/
        login.html
        game.html
        user.html
    data/
        users.db
    utils/
        __init__.py
        game.py
        login.py
    static/
        game.js
        wait.js
```

app.py:

- **home()** - Routed to "/". If the user is logged in, run game(). Otherwise, render_template login.html.
- **matchmaking()** - Routed to "/game". If the user is not logged in, redirect them to "/". If no other users are searching, render game.html with flag waiting on. If

there is another user, instantiate a new server, adding both users to the server, then render game.html with flag waiting off.

- **data()** - Routed to "/data". Used by JS files to get and receive matchmaking + game information from the Python backend.
- **user()** - Routed to "/user/<userid>". Renders user.html with relevant user stats and information from users.db.
- **game()** - "/game" route that will render game.html and allow the user to begin playing the game.

login.html

Presents a page with a form, which will allow users to login to begin using our site. If the user does not have an account, they are able to register on this page.

game.html

If flag waiting is on, run wait.js, then run game.js. Otherwise, just run game.js.

user.html

Takes score info about the user, and formats it into a nice page.

users.db

Will contain SQL databases which will store users with their passwords. Another database will store the user with all their scores.

game.py

class Corporeal - objects that have substance

__contains__(object) - if object is a coord pair and object contains the pair, true, otherwise false

class CircleCollider extends Corporeal - objects that collide like circles

float x - x position of center

float y - y position of center

float r - radius

class LineCollider extends Corporeal - objects that collide like lines
float x1, x2, y1, y2 - coords of points forming line

class Weapon - info about weapon stats, depending on game balance we could have multiple or just 1
float damage - damage/shot
Fraction shotdelay - how many ticks between each shot
float shotspeed - shot velocity

class Player extends CircleCollider - info about one player relevant to the game
int userid - user id
float x - x position
float y - y position
float vx - x velocity
float vy - y velocity
Weapon weapon - weapon
float health - real health, starts at 100

class Map - A map
[LineCollider] walls - (x1, y1, x2, y2) where (x1, y1) is one end of wall, (x2, y2) is other end
(int, int) spawn1 - spawn point of 1 player
(int, int) spawn2 - spawn point of other player

class Instance - One instance of a game
{int:Player} players - playernum:player for each player in game
{int:int} scores - playernum:score
Map map - map

Instance newInstance(userid1, userid2, map) - makes a new game with the two users

Instance.update(userid, post) - takes a post from a specific user and uses that to update game data

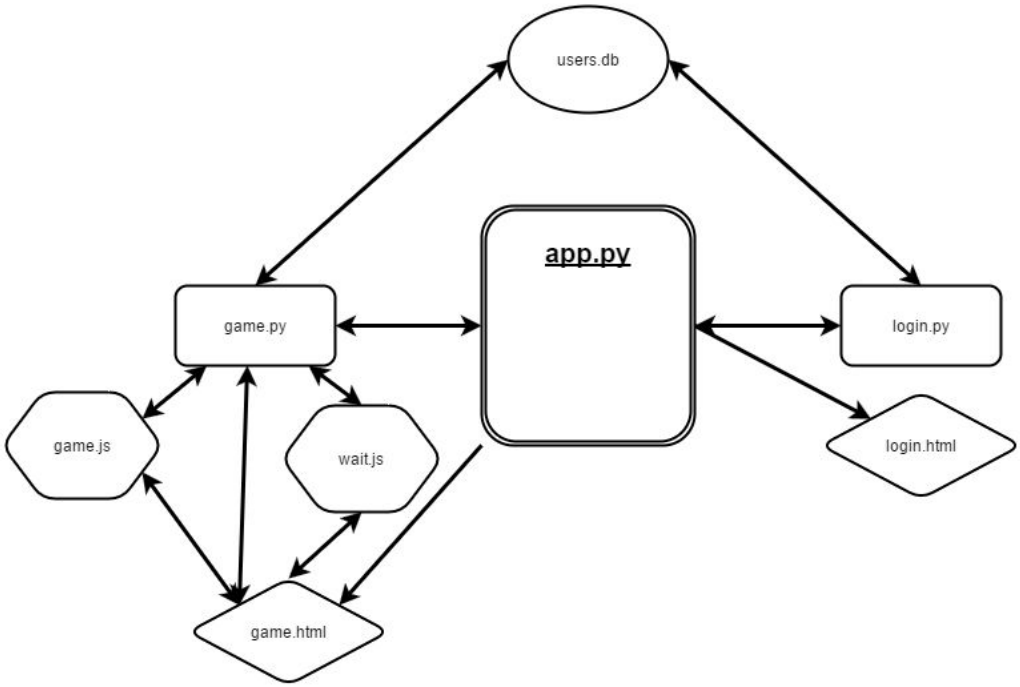
wait.js

Loops a POST request to /data that asks if this user has found a match. When the user finds a match, terminates.

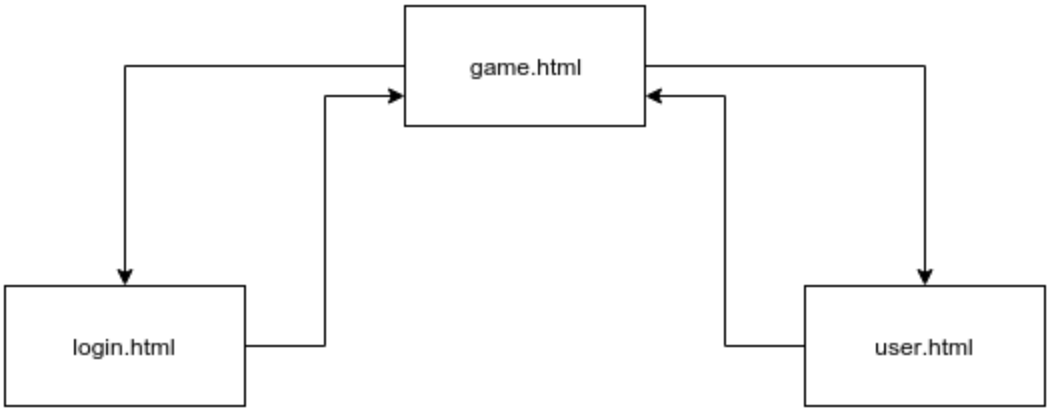
game.js

Loops at a specific frame rate. At the beginning of each game loop, POSTs to /data for the newest data for the current game, and sends current keypresses. Renders game objects based on data received.

Component Map



Site Map



The root of the website is game.html. It contains the game itself along with a way to log in/register (via login.html), and a link to user profile (via user.html).

Database Schema

Accounts

TEXT userid	**TEXT** name	**TEXT** password
User's assigned id	Username	Hashed password

Scores

TEXT userid	**INT** myscore	**INT** otherscore
User's assigned id	User's score in game	Other player's score

Division of Labor

Gabriel- Project Manager + Python backend
Nobel- Python backend

Elias- Main JS backend
Farhan- Frontend + JS backend

Schedule

Date	Task
	Multiplayer test p1 (see if we can have two people online)
	Multiplayer test p2 (see if we can have two people moving + seeing other characters move)
	User database + working login
	Shooting + physics (excluding collision)
	Collision
	Game ending + Not so basic game mechanics
	Matchmaking (with one game instance still)
	Multiple game instances