15-237 Term Project
Colosseum
Nathan Hamal, Zach Matarasso, Dylan Mikus

App Summary

Our app, Colosseum, allows two people to compete head to head in some form of competition while others watch, discuss, and sometimes vote to determine a winner. Losing players are switched out for members from the audience, and the competition continues.

Colosseum is something you can jump into for 5 minutes, and then put down and go about the rest of your day. The fast paced, quick turnover environment lends itself to the mobile platform well. Depending on the type of competition taking place, the target audience can vary greatly. Speed chess can attract users of all ages, whereas a political debate might only attract older users. Anyone with a smartphone and 5 minutes to spare can watch a few rounds of the competition, while desktop users can compete and spectate. Colosseum provides an interactive experience that appeals to fans of any game or hobby that the platform supports.

Technical Challenges

The most technically interesting part of our app is our attempt to make a multiplayer game platform. By having different types of games, Colosseum has the potential to become a platform for spectated 2 player games. Our goal would be to be able to provide an API for communicating game states between the actual game and colosseum client. Our client would handle all other aspects of the game experience besides the gameplay itself. Those features would include an audience chat-room, displaying the correct views to each user, and selecting new competitors whenever a new round starts. Interfacing between a generalised game and the Colosseum client will be a challenging problem.

We also have a few interesting backend problems. We want to be able store games that have been created by other users, keep track of different user's winning statistics, and manage different rooms that won't be persistent. These are all reasonably challenging backend problems, although admittedly the larger challenge will be on the front-end.

Plan

Our primary goal is to set up the basic chat and voting functionality. That is, to set up a space where there are two chat rooms, one for the competitors and one for the spectators, as well as a voting system. Then based on the voting, one user would be thrown into the spectator chat, and a random consenting user would be thrown into the competitor chat. From there, we will work on a webcam battle. Our hope is to implement both of these using the API that anyone else could use to make a game.

After we have two working games, we can start working on creating a backend that will organize rooms and allow for easy placement into rooms. We also want to keep track of some user stats such as win-loss ratio and game preferences. Eventually we might be able to implement team games.

Feature Set

1. text chat: text chat communication between contestants and between spectators
2. video chat: video chat between contestants
3. voting: spectators can vote on which competitor they think is winning or on their preferred competitor
4. competition / game dev platform: we may allow for Javascript to be embedded or
5. demographic information submitted with voting
6. ranking system for contestants
7. team contests

The items are listed in the order that they are scheduled to be developed. Within a week the text chat should be finished, and we should have started on the video chat functionality.By the end of the next week voting and demographic information should be completed.

Week 1
Text and video chat inside of Client

Week 2
Video chat and voting to determine winners

Week 3
Voting and demographic information

Week 4
ranking systems and team contests

Software Stack

1. Javascript: client-side and server-side programmed in Javascript
2. HTML: for client side and maybe server-side tools if we find we need to create any
3. CSS: for client side and maybe server-side tools if we find we need to create any
4. DOM manipulation: used in user text chat (such as user names being links to profile pages)
5. Node.js with Express: the server that will serve all of the information we need it to serve, except for the socket information, which it won't serve, but will instead be served by the socket server.
6. Websockets: server/client communication for text chat and quick communication features. Websockets will also be used for coordinating connections from WebRTC. WebRTC requires another library to perform signaling and handshaking operations, which WebRTC is not responsible for and does not have an API for. If we implement sandboxed games, they can use Websockets for relatively instantaneous communication.

7. JQuery: DOM manipulation, general utility. Will be used to display chat and for visualization transformations.
8. PhoneGap: simplify cross-platform differences for Android and iOS by allowing us to develop native apps inside a webkit.
9. MongoDB: server-side storage of colosseum, competition, and user information
   This will keep track of the rooms, which users are in them, and what is being said in the rooms. Also, all account information, such as profile picture, account name, background information, social security number and achievements.
10. HTML5 video library: if we expand competition functionality to include some kind of video streaming
    a. Entropy Wave: "Serve live video streams provided either by an encoding application or the Entropy Wave E1000 Multi-format Live Encoder. Live streams can be embedded in any web page using either the HTML5 video tag, a custom HTML5 video player, or Flash."
    b. WebRTC: Web RealTime Communication: we have largely settled on using WebRTC. WebRTC uses peer2peer connections, but to put more of the workload on the server, we will have the video streamers stream to the server, and the server will then broadcast the video stream to all of the spectators.

    This will be used to stream video from two contestants to a server, which will then serve that video data to all of the spectators.
11. Embedded Javascript: We will be using iframes to display the game. This will allow us to be a platform for different types of games, and allow our users to create their own games using our API. They will host their version of the game somewhere, and they will communicate with our socket server with relevant game information.
12. D3.js: for visualization of voter statistics. At the end of each round, we will poll all the spectators and their votes will be compiled to determine a winner. At some point, we will take demographic information from the user, and using that, and their votes, we can create graphics that display interesting things about the data set. We can have a line graph of opinion over time, bar charts, and other data visualizations.

13. mongo-express-auth: We will use evan's modules to handle user accounts setup. This will does the work of interfacing sessions with mongo.