Mini-Game Collection Project

Summary

For our project, we originally planned to create a framework of minigames that would work together for a game round. This game would be multiplayer and would be scalable. The idea was to create a main driver that would group players in a lobby and enter them into the game sending them through a series of rounds of minigames. Games would be chosen at random and not repeated. After a specified amount of rounds are completed, a winner is displayed and an overall summary scoreboard.

Github: https://github.com/trladd/CS437\_GameDev\_FinalPrj\_GO3FC

**Game Design Summary**

(For more details and original planning document see the github repo)

This game design plan was to start out in a lobby. In the game lobby, players would choose a game to join and be paired with another player. After the pairing is completed, the game controller loop will kick off. First the game controller will set up basic data connectivity and joining the players. Then it will randomly choose from a list of published games. After choosing from that list of pusblished games, it will direct the players into a minigame which will have varying content. Upon completion of a minigame a game data object will be passed back for later scoring. At this time players will either be directed to a new game or a scoreboard.

**Software Engineering Plan**

When starting this project we first decided on a topic or game that we would like to create. After we had an idea of what we wanted, we started to think about how we would complete this. We decided that it would be best to use JavaScript for a few reasons. JavaScript provides great portability, was familiar to the team, and has just enough power while remaining fairly simple to code.

After choosing what we wanted to do and how we would do it, we started the process. Roles are described below.

Member Roles

|  |  |  |
| --- | --- | --- |
| Trevar Ladd | Madelyn O'Dell | Ernie Yarbrough |
| -Setting up communications and Github repository  -Set up original planning document and layed out structure for ongoing work.  -Created minigame driver and main methods seeking an expandable solution that was easy to communicate with the rest of the project team.  -Exploring different types of multiplayer function and how to implement multiplayer games given peer.js  -Working on more basic minigames  -Minigame integration (incomplete) | -Came up with original minigame plan relating it to a known game called Mario Party  -Contributing to original game planning document  -Brainstorming game ideas  -Building mini-games  -Collaborating with team to integrate created minigames into master branch (incomplete) | -Assisting brainstorming and ‘scrum’ like meetings  -Contributed by aiding in conversation and brainstorming.  -Building mini-games  -Collaborating with team to integrate created minigames into master branch (incomplete) |

**State Transition Diagram**

ScoreBoard

Game Lobby (matchmaking)

Game Controller

(game routing and management)

**User Instructions**

This collection of games was never completed and there were many problems along the way which presented us with a great learning experience. To access this game visit <https://cs.iupui.edu/~trladd/GameDevelopment/FinalPrj/master/menu.html> which houses the basic framework for multiplayer management. Please note that this is not in a completed state and so some content may be missing. Basic logic is there but not all games are included in the paths.

Lessons Learned

* A true back-end for multiplayer games is ideal. While attempting to use Peer.js seemed to be the easy solution in the beginning, it resulted in extra work which contributed to the lack of end-product. While some libraries may seem easy to implement, their lack of power or purpose may present farther issues down the road.
* Multiplayer game logic presents itself in different forms. There is turn based gaming such as checkers, and then other more dynamic multiplayer gaming such as a racing game or shooter game. More dynamic multiplayer will rely on a good network connection and strong back end that will work with the game front end resulting in seamless gameplay.
* Peer.js is oriented toward a turn based solution and programming with it has to be structured towards action based programming rather than traditional game loop programming. This can be harder to think about especially considering that both player one and player two will be looking at the same code and so that code would have to work for both (in a simple game). Peer.js is powerful but meant for messaging and other communication such as video chat. While it is easy to implement, it isn’t meant for a true gaming experience.
* When collaborating with a team, it is important to clearly communicate in the beginning what everyone’s roles and expectations are. It should also be communicated how the project repository and documents will be structured. Communication is key so setting up a group communication method early will benefit.

Roadblocks

* There was very little time to work. By the time we had teams and had a formulated plan, there was not enough time for a project of this scale without additional developers.
* Peer.js is easy to implement but ultimately presented major issues in communicating how it works and the logic behind it.
* Canvas drawing for some reason presented errors to all project members. Although a very simple concept (see here: <https://cs.iupui.edu/~trladd/GameDevelopment/lowLevel/lowLevelGraphics.html>), this was difficult in our environment. Both manual drawing on the canvas as well as using libraries that accessed the canvas had issues. Pulling code from within simple game only farther proved an error because simple game could not draw on the canvas.