Arena Games LLC

Colin Munroe, Liam Clift, Corbin Baker, Alexander Tieman, Matt Clayton

Summer 2017

Introduction

Arena Games LLC is a startup company that is currently developing a card game. Card games require large amounts of playtesting in order to produce a fair and balanced product. Therefore, Arena Games LLC asked us to develop a web application to more efficiently test their game than their current practice of using printed cards that need constant updates. The application should allow for two users to play the game with custom-made decks. Due to the incomplete nature the game is currently in, the program allows for easy modification of all card values. This methodology allows for the program to be adaptive to any changes made to the card game as a whole.

Requirements

- Must be able to play full game
- Must have stopwatch style timer for testing purposes
- Must support multiplayer for two players on separate computers
- Must be able to display card image
- Able to read database and format data into cards
- Board layout that supports gameplay
- Extensive commenting (for employers understanding & later modification)
- Works with Google Chrome

System Architecture

Producing the working web application required multiple pieces working together. Figure 1 shows how the pieces interact in an overall sense, and the following sections will describe more in detail what each part is and how it connects with the others.

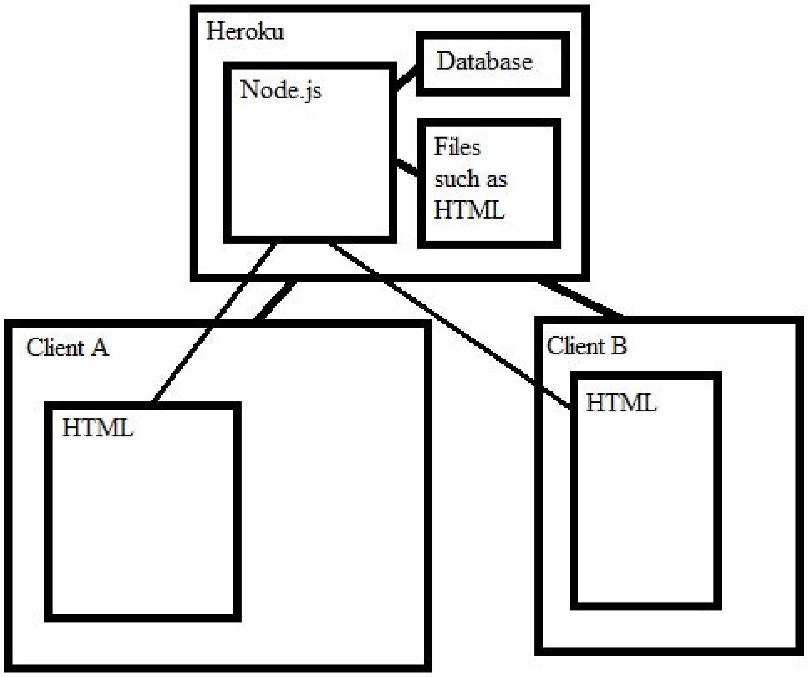

Figure 1: Architecture Overview

Heroku: Heroku is a service for hosting web applications, which allows for deploying applications when completed. This is how users can access the site over the internet, allowing the connection between Heroku servers and the appropriate clients. Heroku also has associated database services and is free for sites that handle less than a limited amount of traffic. This is perfect for our client since they don't plan on ever using the site for anything more than testing with a few handpicked individuals.

Node.js: Node is a javascript runtime environment for using javascript code on the server-side. The server-side javascript is the 'hub' of our code. While Heroku allows the site to be live, Node.js actually creates the possibility for the clients to send data back and forth through the server. This means that Node.js, by giving a place for our code to be executed, and Heroku ,by showing the result of our code to the world, work in tangent to allow connections between clients, the database, and the other files.

HTML Files: The HTML files are the files containing the display information for the pages and some javascript to make the game run. This is the front end of the project, the portion the client sees. It is connected to the back end, or the server, by using socket.io, which allows for information to be relayed back and forth in real time, without the user needing to refresh the page.

Database: The database is where the card data and images are stored. This is connected to the server and ultimately clients by using jQuery commands from the server, which is translated from data taken from the client.

Technical Design

In terms of the technical design, the three main design aspects that can be highlighted are the decisions to use the timer as the main structure point of the program, the selection of the design and layout of the UI, and the choice to use an array for board representation over other data structures.

The timer became an integral portion of the program, even though it was originally suggested as an afterthought because of the requirement of having a live connection for the duration of the game. The timer became the integral way to keep track of where and when the game updates. After the decision on using Node.js as our main server side management method, we ran into the issue of the game requiring the ability to update each player's board in real time.

This became an issue due to the fact that, when the individual board states were to become dissimilar through network error or coding bugs, there would be no way for the game to self-correct. The design needed something that would be consistent for both players and gave a reasonable timeframe as to when the table should update. This function used the same method as the redraw function which was the one that is used whenever a player moves a card from one position to another, as seen in Figure 2.

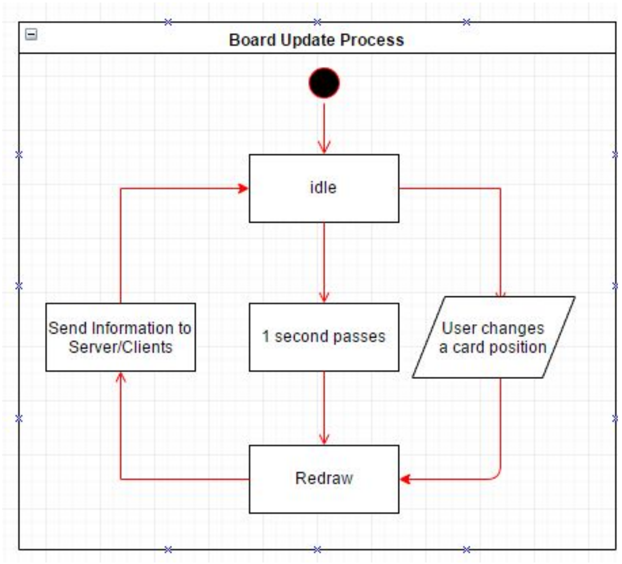
Board Update Process
idle
Send Information to Server/Clients
1 second passes
User changes a card position
Redraw

Figure 2: Board Update Process

The second major design decision that was made was concerning the GUI. The issue that we were encountering was that the constraints of most modern monitors with their respective aspect ratios meant that we must have the board be dynamic in nature and to accommodate all monitor types and styles. This solution was provided through the use of dynamically defined board spaces and a layout that favors a horizontally oriented screen.

This eventually meant that the player's board would be located on the left half of the screen with the opponent's located to the right. In the end, more space would be allocated for larger cards and in turn would allow for ease of viewing. The expanded lateral viewing space allows for an expanded hand space in the bottom of the left half of the screen as seen in Figure 3. However, a shared card area that is accessible by both players as the area would now have to be displayed on both players screens.

This lead to problems with having the board state on both players screens be incorrect and a flickering pattern on the shared area that was later resolved. Eventually this allowed us to have an information area with a phase indication and the timer located in a visually appealing place.

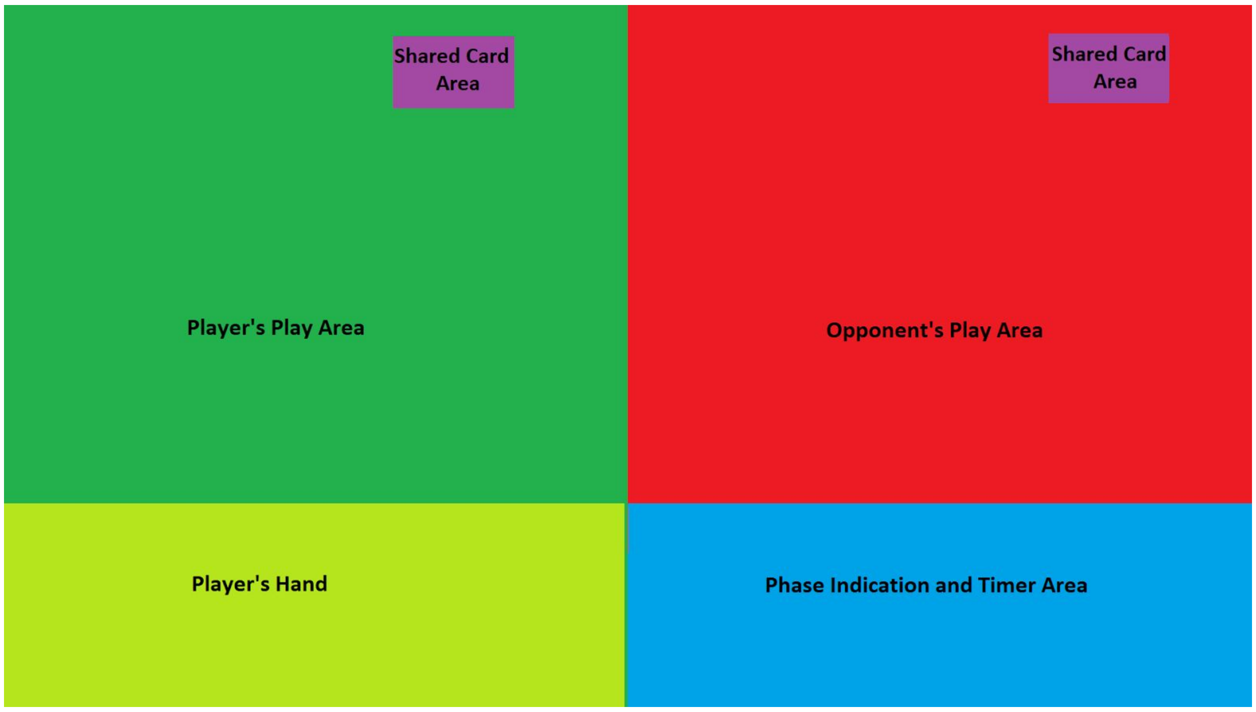

Figure 3: GUI Design

Lastly, the decision to use an array over other data structures for board representation was a carefully discussed decision within the project. When originally deciding on how to represent the board, a few ideas were discussed such as using a hashmap or an array list. In the end however, the resulting usage of an array was decided upon because the board space is a rigid structure and each zone on the board is a defined size. This allowed for us as the developers to decide, when moving cards to specific zones, to only allow a certain number of cards into each zone. This prevents rules violations by overpopulating certain areas and also helps in the design of the user interface because each card within the zone will take up only a certain percent of the area instead of having to compensate for the lack of room for a variable number of card slots in each area. This choice also allowed us to transfer exactly how much information that we wanted to, which in our case was the entire board state.

Design Decisions

Server Over P2P: We decided to use a server style setup as opposed to our original P2P style setup because it would allow us to keep all the code in one place as opposed to needing to have code on each computer. We chose the hosting service Heroku as our means of delivering the product because it was free and achieves the end goal.

Javascript: We chose to use javascript because we wanted to utilize node.js. Node.js would allow us to use the same language for both server and client side. This would make it easier for both our team and our client who will need to be able to work on our code once we are done with it. The utilization of Node.js was also selected because it fit the best with our choice of HTML as a frame for our program. This decision was made so we could make the program a browser application.

Board Layout: Originally we intended to have the board with one player over the other, because that is how many card games are played. However, this became an issue with the fact that monitors, for the most part, are more horizontal than vertical. So instead of trying to squeeze everything and change the creator's layout for the board, we decided to have the boards side by side (Figure 3). This allowed for some other more minor design decisions as well, like where to put the timer and indicators. Since the enemy player's hand was not shown, we were able to put these elements in that area.

PostgreSQL: We chose to use PostgreSQL because it can hold and deliver the information from the database effectively. It was also a convenient option due to the free Heroku add-on as well as most members of the group being familiar with the language already.

Results

Summary of Results:
We were able to get our program to run on an online server that allows the users to play the card game and that follows all the rules that had been outlined within the rules documents and the other rules explained to us by our clients.

Using HTML as the framework, JavaScript as the backend, and a SQL database to hold the various cards, we were able to create a program that would let our client test out their card game with another player. Then using a free online hosting service called Heroku, we were able to put our program online so that our client can access the program from any two computers instead of having the program be tied to an executable. The main test with the client was performed by the team walking the client through the commands and having the client give feedback about the ease of use, look, and functionality.

With the program functioning as intended, our client was able to test his card game. Now he can make adjustments to his game if any changes are need to be made to the game before it is released to the public.

Future Work:

Our main ideas we would have liked to implement were several of the features that we ran out of time before the program was due. Those being:

- Adding a chat-box so players can communicate from a distance
- Having controls being based on mouse interaction rather than keyboard interaction
- A better setup for building decks.

Lessons Learned:

- How to program in JavaScript and HTML. Which objects should be created in the javascript, which objects should be coded in the HTML, what needs to be in the style section in the head and what can go in the Javascript or HTML, and many other more minor situations.
- How to use Node.js and implement socket.io.
- How to program in SQL and more specifically how to connect a database to a node.js through a host.
- How to do basic network interactions, using a hosting service, and having a multi-user, multi-machine program.
- Flexibility within the field of computer science is paramount. Sometimes the research and code you may have written will become irrelevant, but having this realization and flexibility is important.
- Sometimes you have to take pieces from multiple tutorials to learn what you need to learn.