TIMBER: Web-Based Gaming

Mia Belliveau, Mia Blanchard,
Zane Deaton, Samuel Sovereign

8 December 2020

Introduction

TIMBER is a web-based market simulation game that involves buying and selling timber from various regions. There are sets of rules as to how a player can buy, sell, or upgrade factories. The winner is ultimately the player or group with the most profit generated. Our team was responsible for creating the web interface of the game, while the client worked on the back-end and server related features. The web interface includes features such as buttons, hoverables, and the overall visual presentation and ability to parse information from the server onto the webpage.

The end goal of this project was to ensure our interface was user-friendly and visually appealing, as a showcase for the Regis Company's latest software. With that in mind, the biggest challenge was to fit in all the necessary features and simultaneously avoid clutter for an overall intuitive experience for those with less technical knowledge in our field. In addition, our end product was to match the aesthetics and style guidelines given by the client. With respect to the software, our web interface should transfer and interact with information on the backend, including the server data. Since our client was responsible for the mechanics of the game itself, our task was to ensure that we would properly relay and update the information of the game server. For instance, the cash values should change when a region's factory is bought or sold, and the passage of time should be displayed in months at an accelerated rate.

Requirements

High-level description

This program will simulate a virtual economy as a game. Players will be separated into teams and compete against each other in this setting. The overall goal is to display this economic system in simple terms to showcase the technology of Regis Company and to provide the entertainment of playing a game.

Functional Requirements

On the broadest level, TIMBER should handle team-based gameplay, where multiple players join together on a server to compete against each other in groups. In addition, the game should effectively model a simple market economic system by translating the information from the back-end onto the interface. The interface should be graphical, allowing keyboard and mouse inputs, and interacting with online servers.

Non-functional Requirements

While the premise and mechanics of the game's back-end should be simple, the setting itself should be plausible and realistic. In our project, TIMBER takes place worldwide, with 10 regions distributed throughout the map. Visually, there should be a consistent color scheme. To do that, we communicated with our client for the specific color swatches for the design of the webpage. Within the game itself, a passage of time should be properly displayed. After a relatively short amount of time, the months should update in a cycle: We implemented a clock to display this to the player(s).

Definition of Done

Our project is entirely focused on the front-end, which means the user interface should function. The finished product should also include all access to the features of the game both visually and functionally. Since it's web-based the program must be able to be hosted online.

System Architecture

The way that our project was designed, we ended up with three primary components and each of the primary components contain other smaller components. The three primary components are as follows:

Header

The header component consists of three smaller components: Clock Component, Log Component, and Player Profile Component. The header is shown in figure 1. The Clock Component is what we use to show in-game time to each player. In our case time is shown in the form of months from January to December. The log component is what shows players events that take place during the game. These events can vary from random events such as forest fires or player-driven events such as another player is moving into your region. Finally, we have the Player Profile Component which consists of revenue and expenditures as well as total cash the player currently has available to them.

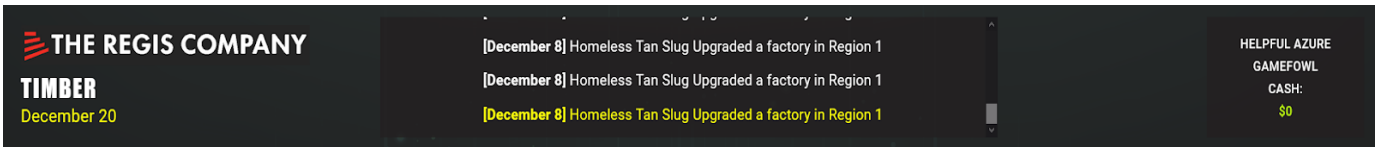

Figure 1 - Header Component

Playable Region

The Playable Region Component consists of ten Region Dot Components, shown in figure 2. Each of these Region Dot Components has a variety of information depending on whether or not the player has a factory in this region. If the player does not have a factory in this region a simple line of text will tell the player that they do not have a factory and they can buy one. If they do have a factory then this will display information such as how much their factory is producing, the current cost of lumber in this area, and will also display a pie chart displaying the percentage of lumber shares that each player has in the region.

Figure 2 - Playable Region Component

Figure 3 - Region Container Component

Region Container

The Region Container Component consists of ten Region Box Components, shown in figure 3 above. Each of these Region Box Components has a button that when clicked will pop up with a box for the player to manipulate their production in. This is the primary place where a player will be playing the game. The box contains several functions such as the ability to buy a factory, upgrade a factory, transfer lumber, sell lumber, and increase or decrease the price of the lumber they wish to sell. The figures below show each component associated with the regions.

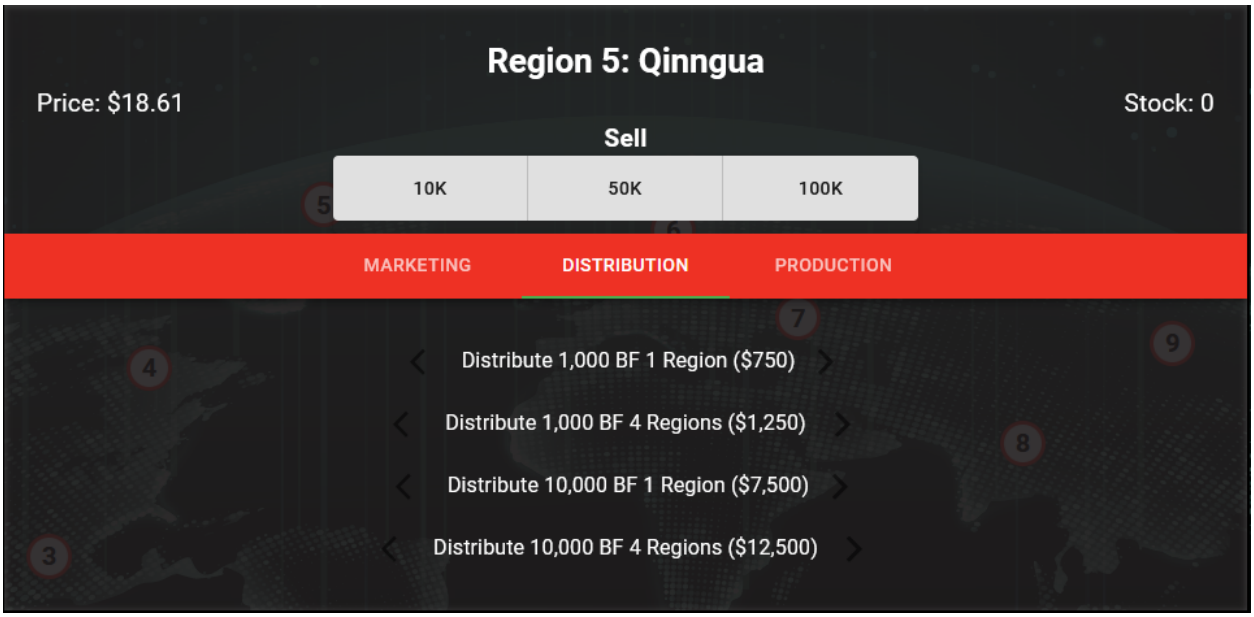

Figure 4 - Region Box with Marketing Tab

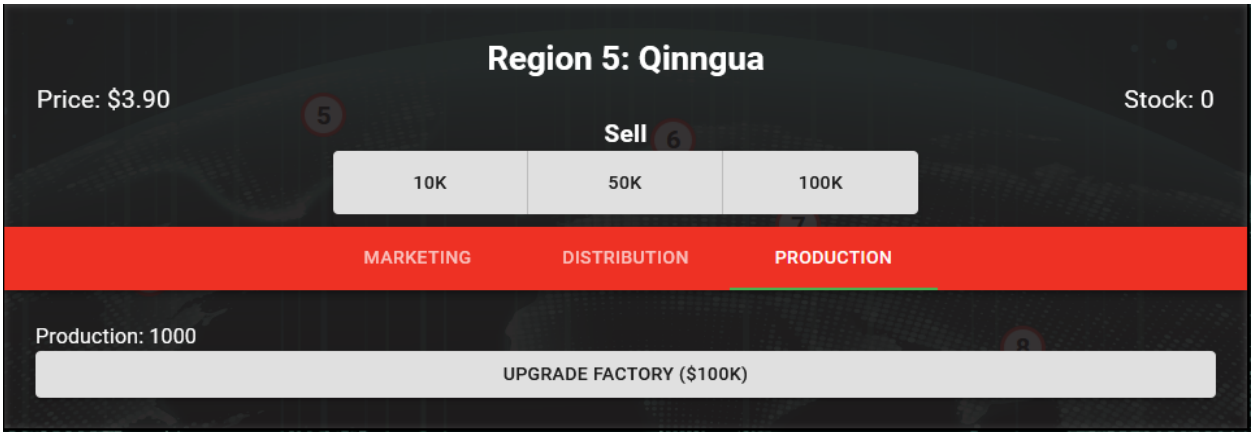

Figure 5 - Region Box with Production Tab

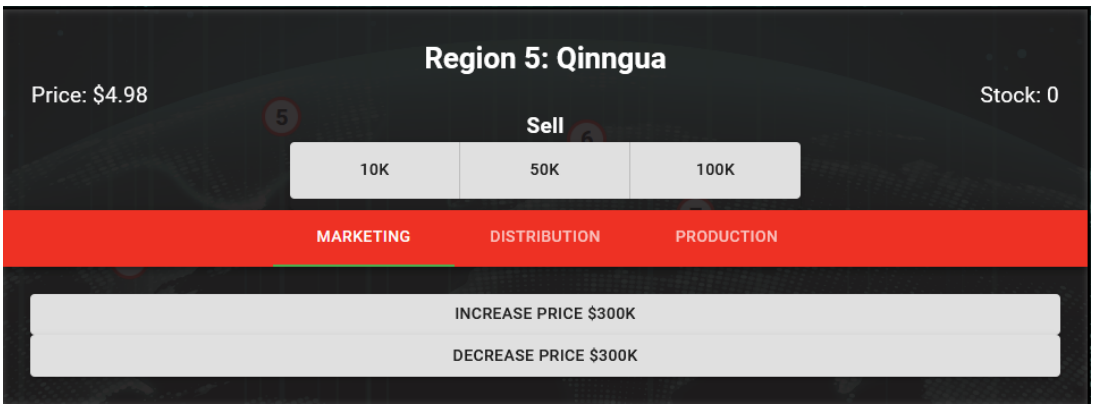

Figure 6 - Region Box with Distribution Tab

Technical Design

Hoverables

Hoverables are features in which a button is moused over, certain code runs if that condition is met For instance, Each dot on the map represents a playable region. When the mouse is over it, a bubble pops up at the upper right corner of the map showing the information about the region relative to the player, such as factories owned

Buttons

Buttons handle events when the mouse is clicked over it. The region container contains buttons for each region that display a popup in the center of the screen when clicked. The popup contains information about the regions and more buttons to manipulate the game. The additional buttons can be used to sell, market, distribute, or produce board feet. The buttons are the primary way to interact with the game.

Clock

The clock displays an accelerated passage of months in a year. Near the top left corner of the web page displays text that changes within several seconds to reflect a change in the month. This time comes from the timing of the server itself. Within the frontend, when the month is changed is based on a switch statement based on a modulo calculation with 12 from the raw server time given.

Modals

A modal is a window that pops up when an item is clicked and closes when the mouse clicks at a space outside the window, and the rest of the webpage is slightly obscured when this window is up. Each region button along the bottom of the screen prompts a modal with the information and options to manipulate values for a given region. This allows for a more spacious layout of our interface since only one modal is allowed to be up per region on a button click.

Game Implementation

The game is first initialized on the server-side game.js. This file calls two other server files, server.js and region.js, to initialize the state of the server. server.js authenticates players and handles events (such as purchases) to be relayed to the game. region.js initializes the state of each region for each individual player and relays any information (price, stock, etc.) to the game. When a player loads the web-page on a browser, a separate, client-side file named game.js is called. This loads the information from the server and passes relevant information to each component. The sub-components then process the information and display it on the web-page. Below is a figure demonstrating the communication between each file and component.

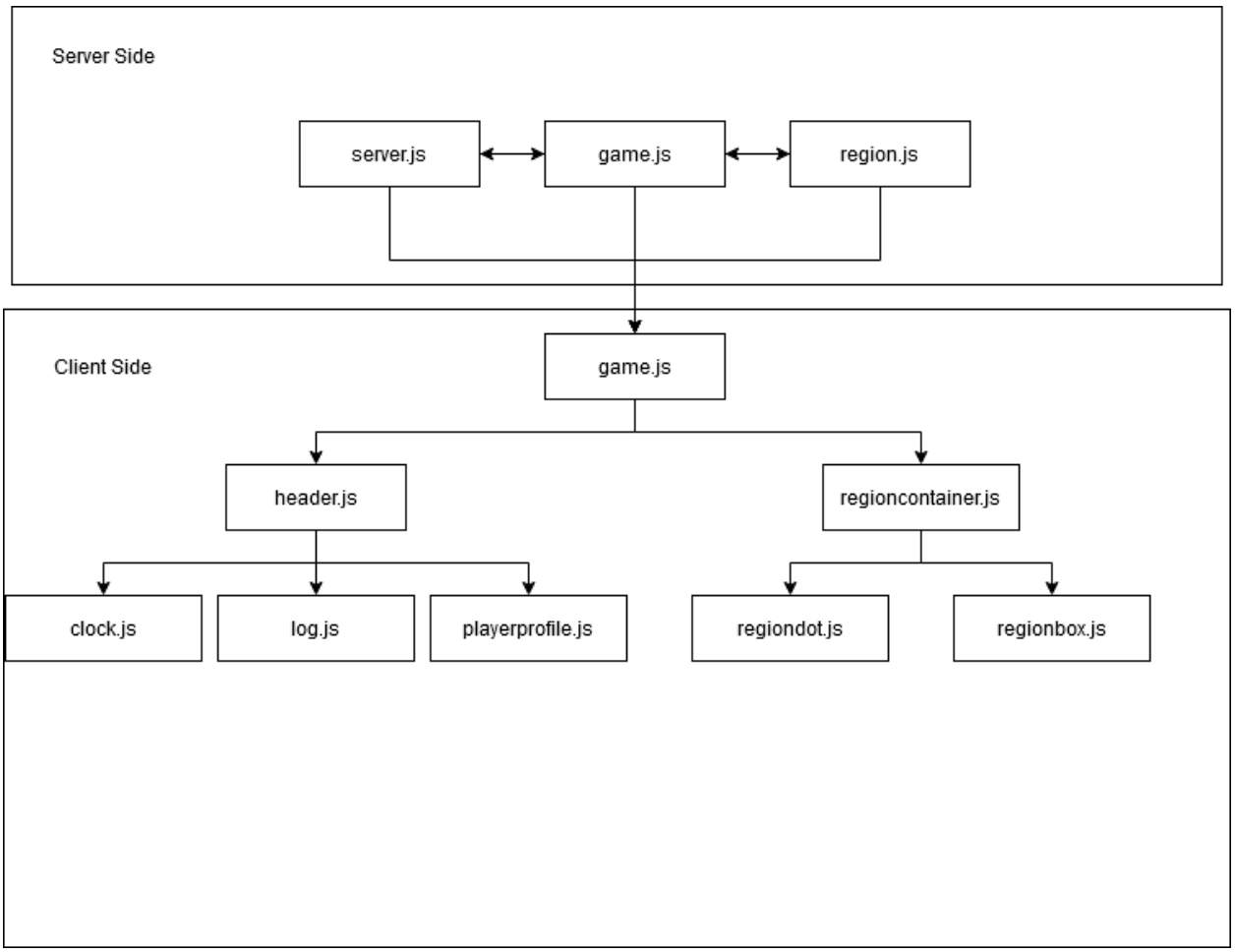

Figure 7- The implementation of the game. Each component relays information to other components to create what a player sees on the browser.

React

React JS is a JavaScript library used to style and build interactive web elements on a page. React is a front-end library, it runs on the browser and determines the UX elements on the page. The React framework is guaranteed to result in better performance, compared to other frameworks, since React aims to update the DOM, document object model, as few times as possible causing apps to load faster and deliver a better user experience. Another popular feature of React is that it gives developers the ability to work with a virtual browser rather than the real browser, which oftentimes is friendlier to develop on.

Problems Encountered

As we were developing the front-end, the pace of back-end development was slower. We were adding features faster than functions could be provided from the back-end. This slowed the project down as we could not test or implement front-end components since some were reliant on these back-end functions. To overcome this, a hard-coded file containing region data was created. This allowed us to test component functionality using this region file and, when the back-end features were implemented, we were able to easily transfer the calls from the hard-coded file to the actual back-end region data.

Quality Assurance

Selenium

Selenium is an API for testing web-based applications. We used Python with Selenium to test our website automatically. It works like other unit testing libraries/APIs, such as JUnit testing in Java. We have written test cases for each element on the web page to be sure that they work as intended and the same across different browsers. For instance, the tests ensure features like hoverables (seeing if items pop up when the mouse is over) work correctly and server configurations are recognized so that information is being properly translated. Each testing event would be noted in an Event Log to show the results of the tests for each unit. A very important aspect of our Selenium testing is it ensured values update when our front-end interface interacts with the back-end data. The Buy and Sell feature especially needed this testing to ensure our buy and sell buttons properly update the money values in the server. In addition, the map elements should change depending on the transfer of ownership between the region.

Git

By using git, we had access to a lot of useful tools that help to ensure quality software. First and foremost, git gave us access to version control. This means that if someone made a change in code that messed something up, it was easy to go back to a previous version that was working. Git also allowed for easy code reviews by comparing the git difference between our branches and master to see what exactly was changed. Another benefit of git is that it allowed us to develop in more bite-sized components. We made a branch for each front end component and merged those into the master branch as we went. This allowed for a slow and steady approach to code development that resulted in more quality code in the end.

Weekly Meetings with client

We met with the client weekly to discuss the changes we've made to the project. Throughout these client meetings, we received feedback from our client for any changes they would like us to make to ensure we are meeting their expectations. Between meetings, our client would often send styling updates for us to implement and adjust accordingly.

Code Reviews

For each major ticket, no matter who did it, we did in-depth code reviews after the person who was working on their ticket said they were finished. This allowed us to have multiple eyes on each piece of code to make sure we were making the best code possible. Without this, there would have been a high likelihood of bugs and crashes popping up that could have easily been prevented by a code review. Also, we made sure to communicate with one another to ask technical questions for coding, as this has helped us learn from one another in building this project.

Jira

Jira is a software for development teams to keep track of their progress for projects and make sure they can meet goals. We created new sprints and epochs as we worked on our project. We would assign tickets to each team member to implement new features, resolve bugs, or rework code. Each ticket was assigned point values in order to track our team's velocity. We also assigned roles to each team member to divide tasks to people who were best suited to them.

Results

We have a functioning web page for our game application. We were able to meet the styling requirements for our application. We implemented all of the required buttons and images for the game to display properly. Some functional requirements were not able to be implemented in time due to the client's back-end limitations. Our client was responsible for coding the functions that our buttons would be attached to, but there were some delays with getting those functions. We did not receive all of the functions until the last week of the project. As far as future works go, we have made it easy to attach the new functionality to the existing features. We have also tested our application to ensure that there are no front-end errors with our current code. We have a test suite built for the website using Selenium which allows us to test our different web page components to make sure everything is working properly.

Overall, this project has been a great learning experience. We learned that some CSS features do not work in different browsers and fallbacks need to be implemented so that the website can display properly on any device. We also needed to consider what the earliest version of a browser we would support would be. A good example of this is with the calc() function. Firefox did not support the function until version 57, which was released in 2017. We had to make the decision whether or not to implement a fallback for users running an old version of Firefox. Ultimately, we decided it wasn't necessary for the majority of users. We also ran into issues with sizing objects with CSS. We wanted to use percent values for sizing so it would adjust itself to the size of the window automatically. However, the size is measured relative to its parent object. There were times where percent values would not work at all because we never set the width and height of its parent. We learned to always check the width and height of the parent when sizing objects. We also learned some interesting ways to test our code using Selenium. With Selenium testing, we needed to learn how to intertwine two programming languages to test our features. We used a Python script to check that our functions and variables were working correctly in our Javascript code.

Appendices

Installation:

- Npm: builds the code environment
- React JS

Development Environment

- Visual Studio Code: GUI application for programming our code.
- Git: version control
- Node JS
- Inspect element
- Client communication

Modeling Technique

- Webpage wireframe: a general visual design of the website's layout and the features to be included.

Coding Conventions

- Consistent coding style: comments, and formatting of the *.js files