UNSCRAMBLE IT: A Word Unscramble Game

Contributors:

- Isaac Kim
- Arpita Kumari
- Neil Thimmaiah
- Lydia Tse

TABLE OF CONTENTS

Overview

The purpose of this project is to create a multi-player unscrambling the word game. In order to build this game with multiple players, the code has been implemented with Socket.io. This open source framework allows to construct a relationship between the clients and the server. To make the game more appealing, along with efficient GUI development, the look and feel of the game is incorporated using CSS Styling. The backend component is programmed through Node.js.

This word scrambling game can be played by multiple age groups. The intention behind choosing this game was to make something educational, as well as fun and interactive. We have tried to make this game user-friendly so that the clients are satisfied and gain more from a personalized experience.

The game starts off with prompting the user to enter in their username for the game. This brings in the personalized touch mentioned before. As soon as four players connect, the game begins. A timer is displayed on each client's screen for 60 seconds. When one client guesses the word, they score a point and everyone goes onto the next word. Throughout this process, clients are rewarded for their victory, are informed if they need to try again to guess the word. The correct word is also displayed when the client is unable to guess the accurate spelling. At the end of the game, every user gets to the results page where they can see where they stand with respect to the other players in the game. Multiple games can be played at once with this implementation.

Choice of Design

The backend of the code is done mainly through Node.js, and the frontend is done using CSS Styling with html. The server/client framework used is Socket.io, and prototypes for the game is established using Figma.

The goal has been to create an educational and pleasant experience for the user. Thus, the game runs efficiently with smooth transition between different components of the game. The data is transmitted in the form of JSON files via the emit and broadcast methods.

Entities

Server

- Purpose: To send client information, receive information, and game logic is all taken care by the server. New clients are able to establish connection, and the words dictionary are also is in this file.
- Methods:
    - removeFromQueue(clientID)
    - startNewGame()
    - scrambleTheWord(originalWord)
    - getRandomWord()
    - sendToGamePlayers(gameResults, message, objToSend)

Client

- Purpose: To create functions required for the client in order to play a fully functioning game. The timer implementation and the scoreboard are both in this file as well.
- Methods:
    - addEvent(object, eventName, functName, cap)
    - init()
    - showModal(modal)
    - showScoreBoard(gameResults)
    - resetTimer(timer)
    - resetTimeout(timeout)

Styling

- Purpose: To create a pleasant experience for the user, and makes the look and feel of the game appealing. This included using components such as textboxes, visuals, and buttons to make the game interactive.

Benefits, Assumptions, Risks/Issues

Benefits

- Unlike desktop applications web applications can be accessed from anywhere at anytime, from any PC.
- The user gets to be in control of the game from their computer allowing them to customize the experience as they wish.
- Establishing the server-client connection is more efficient in Node.js.
- No downloading or installation required, one internet connection is all we need to play!
- A great learning experience for the contributors for creating a web application for the first time

Assumptions

- Has access to internet connection
- Has a modern browser
- Is familiar with the English Language

Risks/Issues

There were some issues along the way we encountered:

- Getting used to Node.js: understanding how we can incorporate the beneficial components, and learning the language
- Establishing the server-client connection so that multiple clients can join the the game, multiple games can be played at once, sending data from server to clients and vice-versa
- Randomizing the scrambling of the words so the user can get to guess a different spelling each time
- Incorporating the timer component: making it start, pause, end and reset at correct times in the game
- Connecting all the pieces together to make the flow of the game smooth
- Implementing game logic with constant score updates among the clients

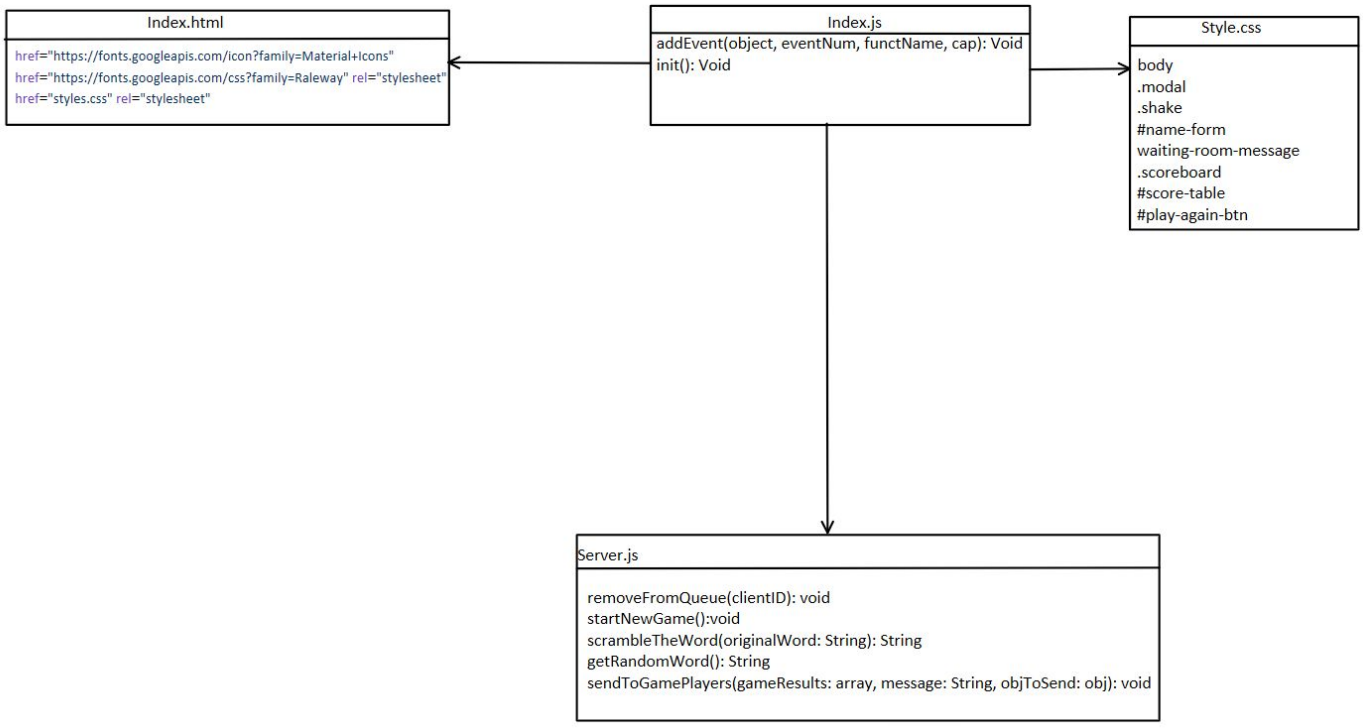
Testing Components

The code has been tested through every step of the way. Once one component was done, we made sure the functionality kept improving. Some key cases we tested for include:

- Does the game update accurately when:
    - tries to join a game?
    - tries to quit the game or play again?
    - tries to enter in the spelling of the word
        - ~ What happens when they enter in the correct spelling
        - ~ What happens when they enter an incorrect spelling
- Does the server generate a different word in each round of the game?
- Do the buttons enable the user to have an interactive experience?
- Does the timer pause accurately, redirects to the results page when the sixty seconds is over?
- Does the game flow naturally with the game logic?

\

UML Diagram


Index.html
href="https://fonts.googleapis.com/icon?family=Material+Icons"
href="https://fonts.googleapis.com/css?family=Raleway" rel="stylesheet"
href="styles.css" rel="stylesheet"
Index.js
addEvent(object, eventNum, functName, cap): Void
init(): Void
Style.css
body
.modal
.shake
#name-form
waiting-room-message
.scoreboard
#score-table
#play-again-btn
Server.js
removeFromQueue(clientID): void
startNewGame():void
scrambleTheWord(originalWord: String): String
getRandomWord(): String
sendToGamePlayers(gameResults: array, message: String, objToSend: obj): void

Activity Diagram