Wheel of Fortune

Version 1.0.0

Preface

Textbook Description: This defines the expected readership of the document and describes its version history, including a rationale for the creation of a new version and a summary of the changes made in each version

Information for Contributors

If you are contributing to this document please add your name under the 'Acknowledgements' section of the Preface.

Links for the project process and creation must be added under the 'Other Documentation' section of this document.

New terminology must be added in the 'Glossary' section along with a description and a possible link for reference.

Use Cases and C4 Diagrams must be added under the 'User Requirements' section in the appropriate place.

Any information added into this document from outside resources must be added under the 'References' section of the Introduction.

About this Document

This document is intended to provide a purpose and specification of the creation of the game Wheel of Fortune. This document was created as a part of a Software Engineering class project at Carroll University. The contents of this document are for practice purposes only and may not be up to standard. Provided is Version 1.0.0 unrevised.

Intended Audience

This document is intended for students in CSC440 who are responsible for game creation and maintenance.

Text Conventions

New to this Release

Version 1.0.0 has no new revisions at this time.

Acknowledgments

Team Members:
Grace Forciea
Adam Nuxoll

Dalton Hein
Jonathan Kenyon

<u>Other Documentation</u>
Getting Started:
Integration Overview:
Online Resources:

Introduction

Textbook Description: This describes the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.

1.1 Purpose of the system (Project Description)

Description: Wheel of Fortune has been a popular televised game show for many years. The game is to solve a word puzzle by the players. Initially, the puzzle is presented as a series of blank spots, one per letter of the puzzle. Punctuation, such as spaces, commas, etc., are displayed openly, in their own spaces, and do not need to be guessed during gameplay.

Players fill in the puzzle in one of two ways. For consonants, players spin a large wheel made up of spaces of varying monetary amounts, along with some "bad" spaces, such as "Lose a Turn" and "Bankrupt". If the spin results in a monetary amount, the player must guess a consonant. If the guessed letter exists in the puzzle, all instances of the letter are displayed, and the player is awarded an amount of money equal to: the number of instances of the letter times the monetary amount resulting from the wheel spin.

Once the player chooses to spin the wheel, they must guess a consonant - they may not "change their mind" and decide instead to buy a vowel or solve the puzzle. For vowels, the player does NOT spin the wheel. Instead, the player announces they wish to "buy a vowel". A set amount (say $250, though it can be a different amount depending on the cost of the vowel) is deducted from their account and the player selects a vowel (vowels can't be bought if the player has less than the required cost of a vowel). If the vowel is in the puzzle, all instances are displayed.

No money is awarded for illuminating vowels, and the cost of buying a vowel is the same regardless of the number of instances of the vowel illuminated. Once a player chooses to buy a vowel, they must guess a vowel – they may not "change" their mind and decide instead to spin the wheel or solve the puzzle. Regardless of consonant or vowel, if the letter chosen by the player is in the puzzle, the player's turn continues, otherwise, the player's turn ends, and play moves to the next player.

If the player spins the wheel and lands on the "Lose A Turn" space, the player is not able to guess a letter, and play advances to the next player. If the player's spin lands on "Bankrupt", any money the player has accumulated is lost (their money score is set to 0) and play advances to the next player. At any time, the active player may choose to solve the puzzle rather than spinning or buying a vowel. The player is given the chance to say what they believe the full puzzle is. The player must type a guess. The guess typed into a text field must match the hidden sentence (the puzzle) exactly including punctuation and white spaces except for case differences.

If correct, the player wins the full amount of money they have accumulated, and all other players win nothing, regardless of how many letters they have guessed correctly and how much money they have accumulated. If the guess is incorrect, the player loses their turn, and play advances to the next player. If players try to solve the puzzle, they must make a guess – they may not "change their mind" and spin the wheel or buy a vowel instead.[10:24 PM]

What to Do? The game description above was mostly adapted from here: https://eecs285.github.io/p3-wheel/. The site also included a demo. However, you might find other similar attempts over the Web; some may even provide the code. You may borrow some ideas from the references, but you specify, design, and construct an independent game product. You can use Java, C#, JavaScript, or even React to code the game (the latter two would allow you to play the game with a browser).

1.2 Scope of the product

Development Tools:

- Visual Studios Code - Source code editor
- Javascript - Programming Language
- HTML - Programming Language
- React - JavaScript library for building user interfaces
- Gatsby - Static-site generation web framework

Component Creation:

1.3 References

https://www.browserstack.com/guide/coding-standards-best-practices
https://www.datree.io/resources/github-best-practices

Glossary

Textbook Description: This defines the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.

Term:	**Description:**
Static vs client-based	Content that is static won't change or update. Examples of content that would be client-based and not static are live chats, email clients, or other components that need to update based on a user's interaction.
Static query	Used to retrieve data from a data layer
OOM (Out of Memory)	A type of risk that occurs if the memory of the program exceeds the amount of storage necessary
Acceptable Failure Rate	The allocation in which a project should not work
Git push	Allows you to add changes to the repository
commit	Commits changes to the repository
.gitignore	Specifies intentionally untracked files that Git should ignore
Master branch	The main branch of the repository
Git client	Keeps track of the initial configuration with a users name and email address

User Requirements

Textbook Description: Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.

3.1 Nonfunctional System Requirements

Product Requirements: (must be converted to natural language)

General description: The product is a web front-end application, to be developed using a front-end web technology such as JavaScript and React. We chose this route because most of us have not worked with JavaScript or React so it is good experience.

Data handling: Game phrases are stored in arrays.

Assumptions**:** players are on the same computer.

When a user loads the website page the data should already be available. This project is static-based and will require compiling pages at build time. A static query may be required to retrieve data from a stored database.

Memory: below 150-200Mb to reduce OOM (Out of Memory Risks)

Acceptable Failure Rate: NA

Security: NA

Usability: This website is intended for use by an average customer. A user-friendly interface is required.

Development Process: Development will be in Gatsby using a combination of Javascript and HTML (React), CSS, and the potential use of C# for backend development.

HTML: A markup language that every web browser is able to understand. It stands for HyperText Markup Language. You use HTML to add structure to your web content, defining things like headings, paragraphs, and more.

CSS: A presentational language used to style the appearance of your web content (fonts, colors, layout, etc). It stands for Cascading Style Sheets.

JavaScript: A programming language that lets you make your web content dynamic and interactive.
The **command line**: The command line is a text-based interface used to run commands on your computer. You'll also often see it referred to as the **terminal**. In this tutorial, we'll use both interchangeably.

React: A code library (built with JavaScript) for building user interfaces. It's the framework that Gatsby uses to build pages and structure content.

GraphQL: A query language that allows you to pull data into your website. It's the interface that Gatsby uses for managing site data.

Development Environment:

Node.js: Node.js is an environment that can run JavaScript code outside of a web browser.

Git: Git is a free and open-source distributed version control system. It's a tool that helps you save different versions of your code. It also helps teammates work together on the same codebase at the same time.

GitHub: GitHub, Inc. is a provider of Internet hosting for software development and version control using Git.

Visual Studio: Visual Studio Code (also called VS Code, for short) is a popular code editor that you can use to write code for your project.

Gatsby CLI: The Gatsby command-line interface (CLI) is a tool that lets you quickly create new Gatsby-powered sites and run commands for developing Gatsby sites.

Environmental Requirements: hosted on Gatsby and displayed as a website to users.

Organizational Requirements:

System is organized in components. One component per file. Components can be exported and imported to other components.

How it will be used: This game is intended as a school project. However, a user should be able to play Wheel of Fortune using a spinning wheel component, numbers, and number updates to track their progress, and a display board for the potential words a player can guess.

External Requirements: NA

3.3 Use Cases

(Complete game of Wheel-of-fortune)

Use Case Overview

Description	**The use case shows possible plays one contestant can make in a single term.**
Actor(s)	**A contestant or player**
Pre-Conditions	**Some characters in a phrase are displayed on phrase board**
Post-Conditions	**Same as precondition, except more characters could be displayed**
Trigger	**Game has been initiated**

Main Flow

<Enter the Main flow of events. i.e. The steps narrating/illustrating the interaction between Actors and the System. Describe Actor's actions/stimuli and how the system responds to those stimuli. Describe the 'happy path/day' scenario, meaning the straight and simple path where everything goes 'right' and enables the primary actor to accomplish his or her goal. Main flow/path should always end with a success end condition.>

Main	
1	**Player enters an input to spin the wheel**
2	**System simulates wheel spin and stop.**

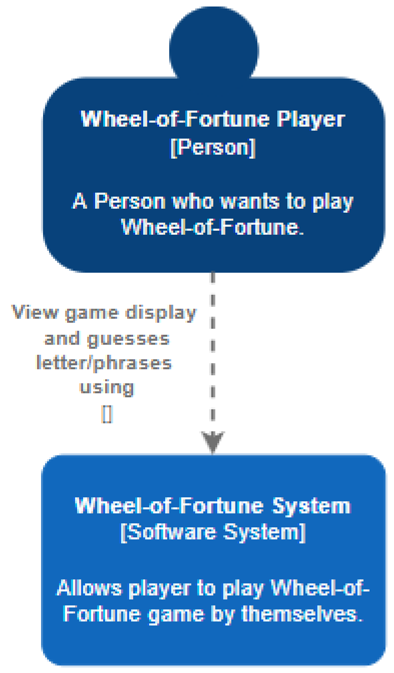
3	**Player enters a letter (to make a guess)**
4	**System determines the letter is in the phrase and unrevealed, and then displays all instances of letters on the phrase-board and updates the amount of cash awarded: existing amount + cash amount shown on wheel.**
5	**Repeat steps 1-4 until** **a) System has displayed all characters of the phrase and declares the player the winner.** **OR** **b) System determines the letter is not in the phrase.** **c) Player enters a phrase, and the system determines the phrase is correct, and declares the player the winner.** **USE CASE ENDS**

Alternate Flows

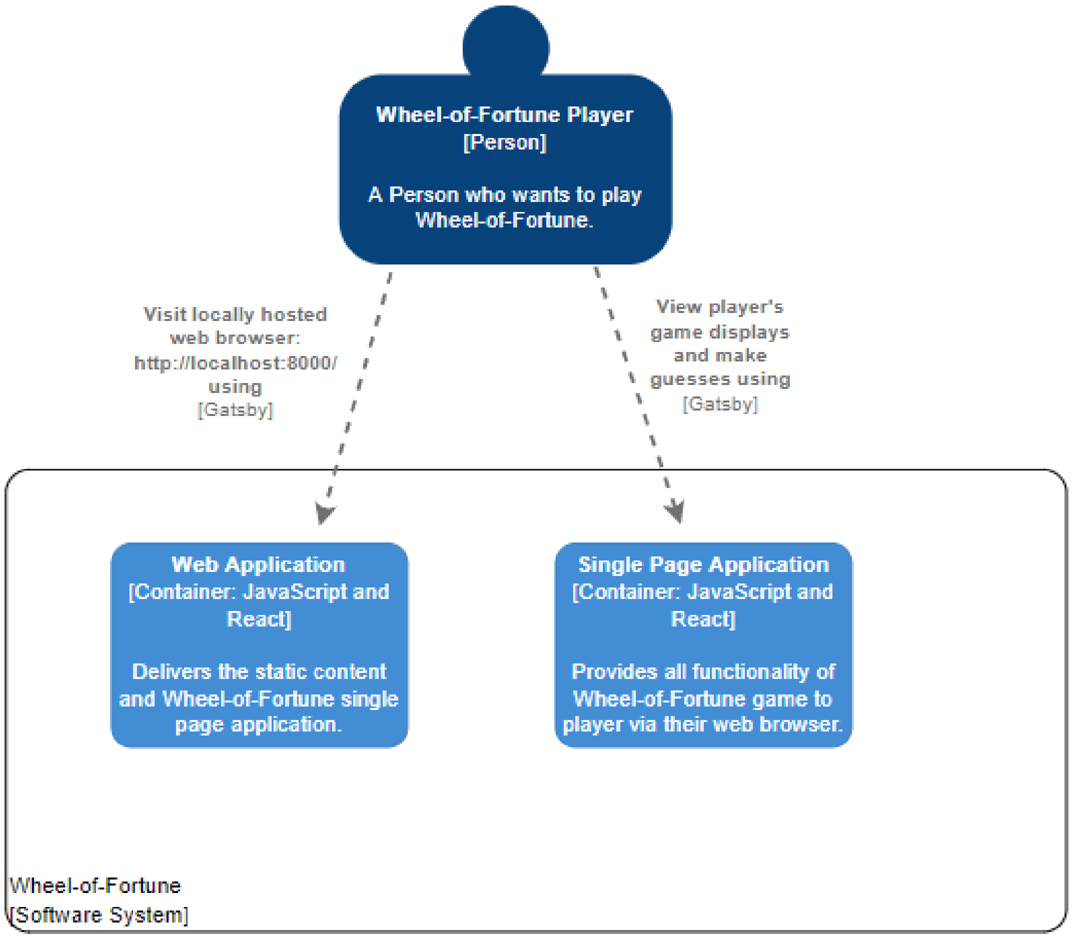
1a	**If player chooses to buy a vowel**
1	**Player enters a vowel**
2	**System deducts the cost for vowel from the player's award amount.**
3	**System displays all instances of vowel if vowel is in phrase and unrevealed.**
4	**Proceed to step 1**
3a	**Wheel lands on 'bankrupt' Space**
1	**System Takes all money away from player that spun wheel**
2	**proceed to main flow step 1**
3b	**Wheel lands on 'lose a turn' space**
1	**System skips the turn of player that spun wheel**
2	**proceed to main flow step 1**

3.4 C4 Diagrams

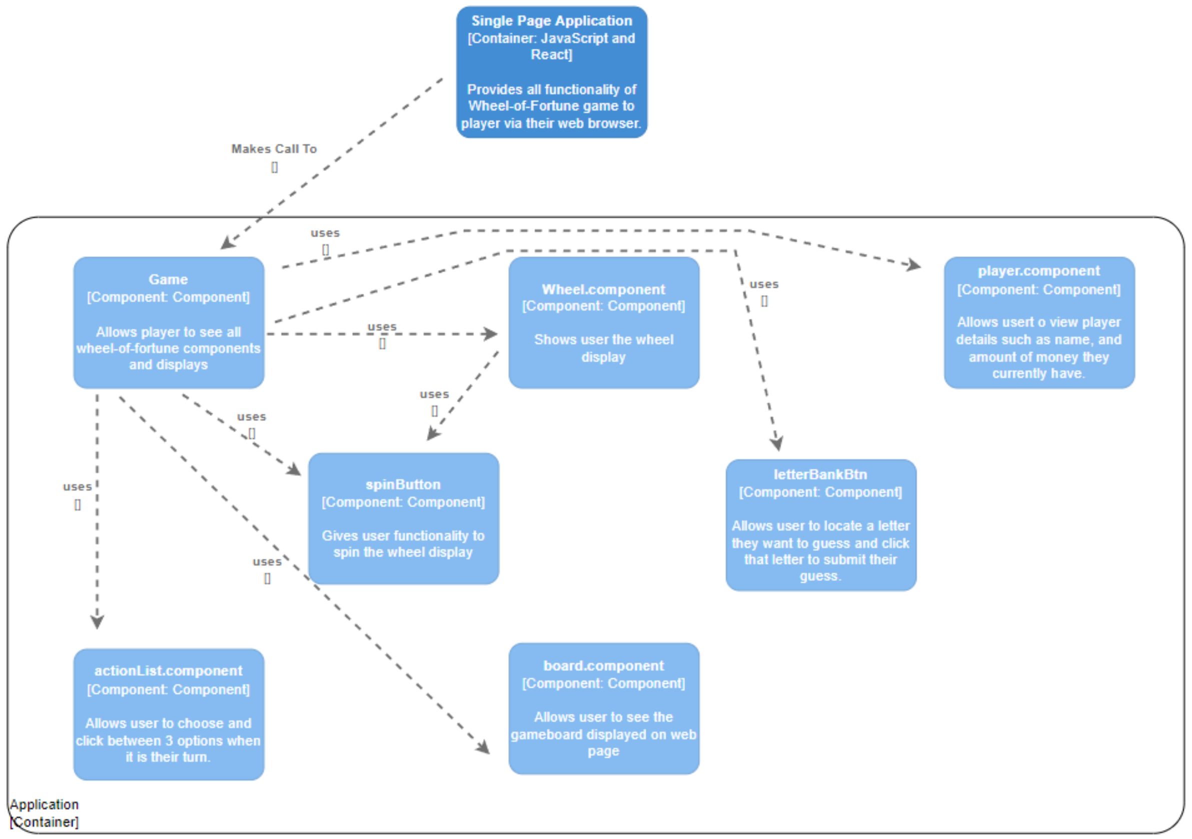
Level 1:


Wheel-of-Fortune Player
[Person]
A Person who wants to play Wheel-of-Fortune.
View game display and guesses letter/phrases using
[]
Wheel-of-Fortune System
[Software System]
Allows player to play Wheel-of-Fortune game by themselves.

Level 2:


Wheel-of-Fortune Player
[Person]
A Person who wants to play Wheel-of-Fortune.
Visit locally hosted web browser: http://localhost:8000/ using
[Gatsby]
View player's game displays and make guesses using
[Gatsby]
Web Application
[Container: JavaScript and React]
Delivers the static content and Wheel-of-Fortune single page application.
Single Page Application
[Container: JavaScript and React]
Provides all functionality of Wheel-of-Fortune game to player via their web browser.
Wheel-of-Fortune
[Software System]

Level 3:


Single Page Application
[Container: JavaScript and React]
Provides all functionality of Wheel-of-Fortune game to player via their web browser.
Makes Call To
uses
Game
[Component: Component]
Allows player to see all wheel-of-fortune components and displays
Wheel.component
[Component: Component]
Shows user the wheel display
player.component
[Component: Component]
Allows usert o view player details such as name, and amount of money they currently have.
spinButton
[Component: Component]
Gives user functionality to spin the wheel display
letterBankBtn
[Component: Component]
Allows user to locate a letter they want to guess and click that letter to submit their guess.
actionList.component
[Component: Component]
Allows user to choose and click between 3 options when it is their turn.
board.component
[Component: Component]
Allows user to see the gameboard displayed on web page
Application
[Container]

System Architecture

Textbook Description: This chapter presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted

Note: There are many different types of system architecture such as layered, event-driven, microkernel, micro-services, and space-based but for the purpose of this project we will be using a layered architecture which is good for inexperienced developers and projects that need to be developed quickly.

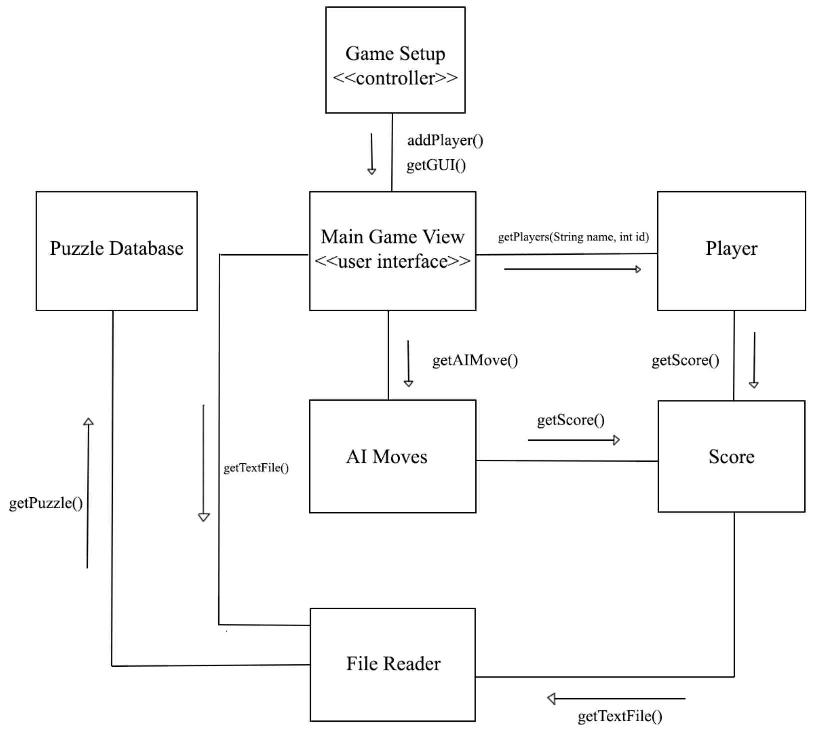
4.1 Layered Approach (Overview):

Presentation Layer: responsible for handling all user interface and browser communication logic

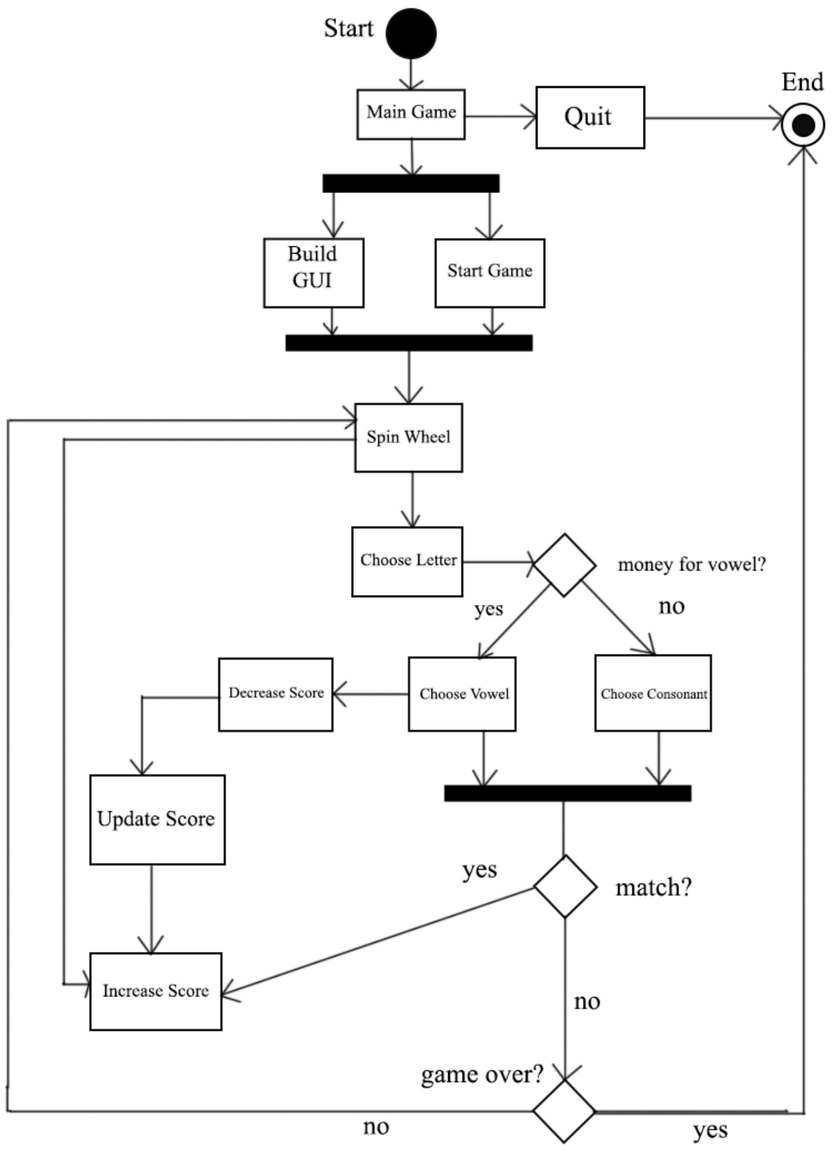
Business/Persistence Layer: responsible for executing specific business rules associated with the request

Database Layer: responsible for containing any saved data from the user or system

4.2 Logical View (Communication Diagram):



4.3 Process View (Activity Diagram):



System Requirements

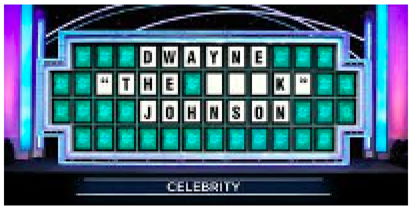
Textbook Description: Here, one should define exactly what is to be implemented. It may be part of the contract between the system buyer and the developers.

General

1. There shall be 3 main components to the UI: The Phrase Board, The Wheel, and The Player Board.

Phrase Board

1. The Phrase board shall have 52 squares, organized in 4 rows: 12 squares each in the top and bottom rows, and 14 each in the middle two rows. The board shall look symmetrical.
2. At game start, all squares shall have a green background.
3. Squares representing characters in a phrase, including punctuation marks, shall have white background.
4. All characters, including punctuation marks, shall be black.
5. Neighboring words shall be separated with a green square or a punctuation mark.
6. A phrase (to be guessed) shall be approximately centered when displayed (hidden or revealed)



Wheel

1. The Wheel contains 24 spaces which when landed on shall either grant a specified amount of money (which is written on the space) on a successful guess action, skip the player's turn, or skip the player's turn and also take away their money. Wheel shall be labeled with Strings in center of each space.
2. System shall spin wheel. When wheel stops spinning, the space that is pointed to by pointer on wheel is the space awarded to the player.

Player Board

1. The Player Board shall display all players in the game and their current money scores. Money scores shall be calculated by summing total game winnings and deductions throughout the game.
2. System shall prompt player with 3 options: spin the wheel to see if they can guess a letter, buy a vowel, or guess the phrase.
3. If a player chooses to buy a vowel, system shall not let player spin the wheel, but will take money equal to the constant set amount for vowels from the player. If the player does not have enough money, they shall not be able to choose to buy a vowel. If the

bought vowel is part of the phrase, system shall display guessed vowel in the white squares that contain that vowel.
4. When a player correctly guesses the phrase, the game shall end and that player is designated the winner. System displays the total winnings of the winner.

System Evolution

Textbook Description: This describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.

The game was created using components in the JavaScript language. Each component has a single task to complete which allows for easier evolution to code and easy maintainability.

<u>Possible Future Design Changes:</u>

- Update system to include a database full of possible phrases.
- Add a database connection to keep track of high scores and player stats.
- Designers can include an option to add CPUs to play against.
- Add an option to "Play Online" with friends or other real players.

Description About Team Activities

Our primary way of communicating throughout this project was via Discord. Discord is a program that allows for group texting chats and ways to communicate ideas to team members. We also created a Trello board in order to divide up tasks and complete weekly goals. After each class we held meetings usually taking 30 minutes. In these meetings we discussed what everyone had completed in the previous week and went over goals for the coming weeks. Everyone on the team completed parts of this document, as well as contributed to the code of the game.

Team Reflection

Overall the teamwork on this project was excellent. After every class everyone was clear on what needs to get done by the next week and everyone followed through on their goals. Communication was excellent on discord, and members of the team helped each other when needed in a timely manner. It was also cool to see that everyone was able to add bits of code to the game to make the game run as described.

How to Use Program

First clone the Repository from GitHub to a local file on the user's computer. Then in that folder, install npm. Next install Gatsby, and Visual Studios Code to Windows computer. Load up the command prompt and type "Gatsby Develop" to run the application. When the web page loads up, player 1 will begin their turn and choose whether to guess a letter, buy a vowel, or guess the phrase. Play continues until the player guesses the phrase or board fills up with letters.