Course: Net-Centric Programming **Course ID:** IT096IU

INTERNATIONAL UNIVERSITY - VIETNAM NATIONAL UNIVERSITY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NCP Group Project

Wonder Words Web Socket Game

NAME	STUDENT ID
Vũ Nhật Duy	ITITIU17047

Instructors: Dr. Le Thanh Son

Academic Year: 5/2023

Table Of Contents

Abstract

This documentation presents the architecture and implementation of a real-time racing game inspired by the popular game show "Wheel of Fortune." The game is designed to be played using a web browser and utilizes Python and WebSocket technology for communication. The objective of the game is to guess a word by revealing its characters within a specified time limit. The game features a server-client model, where the server selects a random word from a JSON file and sends it to all connected players. Unrevealed characters are displayed as hyphens, while spaces and the word description are shown as is. Players take turns making character guesses within a 10-second countdown. If a guess matches one or more characters in the word, the partially revealed word is updated and broadcasted to all players. The guessing player receives points based on the number of appearances of the guessed character in the word. Incorrect guesses result in an error message. The game continues until the word is fully revealed, and players' scores are calculated accordingly. This abstract provides an overview of the game's key features and implementation details to facilitate understanding and further exploration of the documentation.

Chapter 1: Introduction

The purpose of this project is to develop a real-time racing game that simulates the popular game show "Wheel of Fortune." The game will be implemented using Python and WebSocket technology, allowing players to participate using a web browser. The primary goal is to create an engaging and interactive gaming experience where players can test their word-guessing skills in a competitive environment.

1.1. Overview of the Project

The project revolves around a server-client architecture, where a central server manages the game logic and communication with multiple connected clients. The server selects a random word from a JSON file, conceals the characters, and broadcasts it to all players. Players take turns guessing characters within a specified time limit, and the server verifies the guesses, updates the revealed characters, and provides feedback to the players. The game continues until the word is completely revealed, and players' scores are calculated based on their successful guesses.

1.2. Purpose and Goals

The purpose of developing this game is to provide entertainment and challenge to the players. By emulating the "Wheel of Fortune" game show, it aims to recreate the excitement and anticipation of revealing hidden words. The game encourages quick thinking, strategic guessing, and friendly competition among the players.

1.2.1. The main goals of the project include

1. Developing a robust and scalable server-client architecture using WebSocket technology to facilitate real-time communication.
2. Implementing a random word selection mechanism from a JSON file to ensure variety and unpredictability in each game session.
3. Designing an intuitive user interface that displays concealed words, accepts player guesses, and provides real-time feedback.
4. Incorporating a countdown timer to create a sense of urgency and enhance the competitive aspect of the game.
5. Calculating and displaying players' scores based on the number of successful guesses and the frequency of characters in the word.
6. Ensuring the game is playable on various web browsers, providing a seamless and accessible experience for the target audience.

1.2.2. Target Audience

The game targets individuals who enjoy word games, puzzles, and competitive challenges. It is suitable for players of different age groups and can be played individually or in groups. The intuitive web interface makes it accessible to a wide range of users, including casual

gamers, enthusiasts, and those seeking a fun and interactive way to test their word-guessing skills.

Chapter 2: Architecture

The Wonder Words game is built using a client-server architecture, where a central server manages the game logic and facilitates communication with multiple connected clients. The clients interact with the server through a web browser, enabling a user-friendly and accessible gaming experience.

2.1. High-Level System Architecture

The system architecture consists of the following components:

1. **Server**: The server component is responsible for coordinating the game, managing player interactions, and maintaining the game state. It handles tasks such as word selection, concealing and revealing characters, verifying guesses, calculating scores, and broadcasting updates to all connected clients.
2. **Clients**: The clients are web browsers through which players access and participate in the game. Each client interacts with the server to send guesses, receive game updates, and display the game interface to the player.

2.2. Components and Their Responsibilities

1. **Game Manager:** The game manager component is responsible for managing the overall game flow. It handles tasks such as starting the game, tracking player turns, managing timers, and determining when the game ends.
2. **Word Selector:** The word selector component selects a random word from the available word list stored in a JSON file. It ensures that each game session offers a unique word for players to guess.
3. **Guess Verifier:** The guess verifier component validates the guessed characters received from players. It checks if the characters match the hidden word, updates the revealed characters accordingly, and notifies the server of the correctness of the guess.
4. **Score Calculator:** The score calculator component calculates the scores for each player based on their successful guesses. It considers the number of appearances of the guessed character in the word and assigns points accordingly.

2.3. Communication Protocols:

The Wonder Words game primarily uses WebSocket communication protocol, which provides a persistent, bidirectional connection between the server and clients. WebSocket enables real-time, low-latency communication, allowing for instant updates and interaction between the server and clients.

2.4. Server-Client Interaction:

The server and clients interact through WebSocket using a request-response model. The server sends game updates and prompts to the clients, and the clients respond with guesses and acknowledgments. The server broadcasts relevant information, such as the updated revealed word or score changes, to all connected clients to ensure consistent game state across all players' interfaces.

The server-client interaction follows a synchronized turn-based approach, where players take turns guessing characters within a specific time limit. The server manages the turn rotation, starts and stops the timers, and enforces the rules of the game to maintain fairness and consistency.

Additionally, the server uses TCP (Transmission Control Protocol) as the underlying transport protocol for WebSocket communication, ensuring reliable and ordered delivery of messages between the server and clients.

Chapter 3: Game Play

3.1. Overview

The "Wonder Words" game is a real-time racing game that simulates the popular game show "Wheel of Fortune." Players participate in the game using a web browser, and the game is programmed using Python and incorporates WebSocket for communication. The game revolves around guessing words within a specified time limit and earning points based on correct guesses.

3.2. Game Rules and Mechanics

1. **Objective:** The objective of the game is for players to guess a hidden word correctly within a limited number of turns or before the timer runs out.
2. **Gameplay Flow:** The game follows a turn-based system, where players take turns making guesses. On each turn, a player can guess a character from the word or make a guess for the complete word. The game continues until the word is fully revealed.
3. **Turn-based System:** The order of turns is determined either by a predefined sequence based on the order of registration. Each player takes their turn one after another, and the game progresses accordingly.
4. **Guessing Mechanism:** Players can make guesses by selecting a character from the available options. The guessing can not be empty.
5. **Scoring:** Scores are calculated based on the number of guessed characters. The formula is 100 x the appearances of the character in the word.

3.3. Word Selection and Gameplay Process

1. **Word Source:** The list of words is stored in a JSON file format. The game accesses this file to obtain words for the players to guess.
2. **Randomization:** To ensure variety and prevent predictability, the game can randomize the selection of words from the available word list for each game session.
3. **Word Representation and Rendering:** When all players are ready, the word and its description are sent from the server and displayed on the web interfaces.
4. **Hidden Word:** Initially, the word is hidden from the players and represented by hyphens, to indicate the number of letters in the word.
5. **Revealing Mechanism:** As players make correct guesses, the game gradually reveals parts of the word. This could involve replacing the corresponding placeholders with the correct letters or revealing letters one by one.
6. **Turn-Based Gameplay and Timer:** Each player takes turns to guess the character within a timer of 10 seconds. If the guess is correct, the player gets another turn. Otherwise, a message “Wrong Guess!” is displayed and passed the turn to the next player.
7. **Turn Order:** The order of turns is determined based on the order of registration. Each player takes their turn in the specified order.
8. **Timer Mechanism:** A countdown timer of 10 seconds is used to limit the duration of each player's turn. The timer starts at the beginning of a player's turn and continues to count down until it reaches zero. When the timer runs out, the player's turn ends.

Chapter 4: Use Case Diagram

4.1. Use Case Diagram

The following use case diagram depicts the role of the player who participates in the system, understanding how the player interacts with the game.

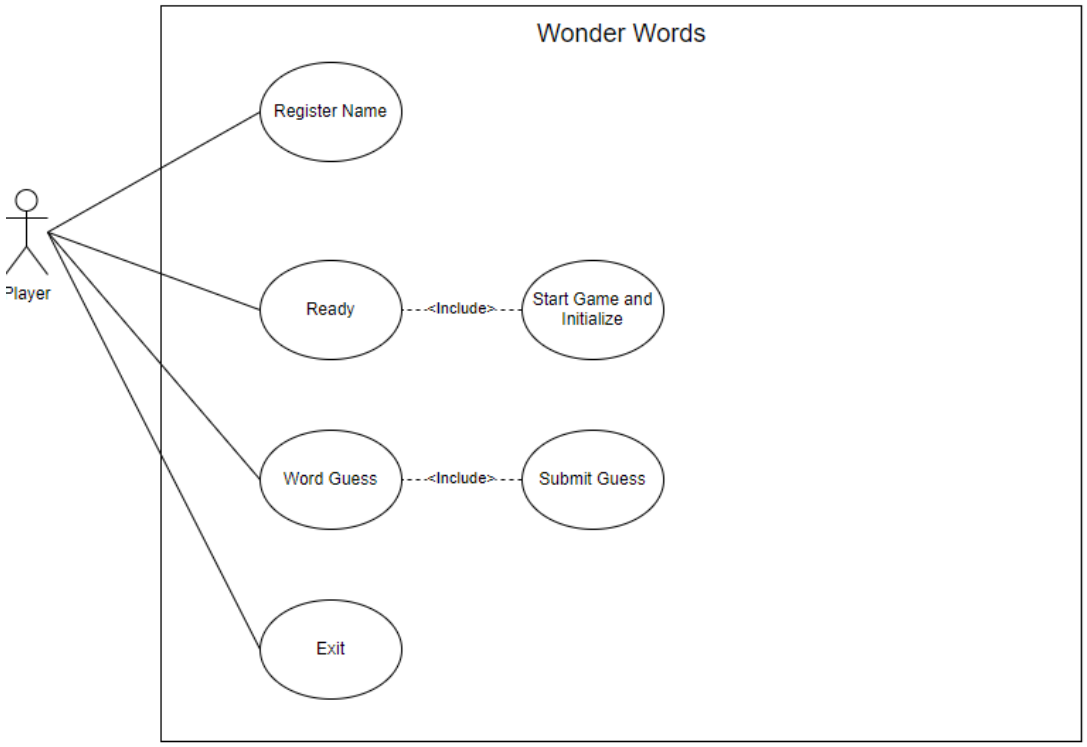

Diagram: Use Case Diagram

4.2. Use Case Description

Use case description is a written description of how users will perform tasks on our system. It outlines, from a user's point of view, a system's behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled. Use case descriptions depict all possible cases that can happen within an use case diagram.

4.2.1. Use Case: Register Name

Description: This use case allows the player to register their name in the game. The player provides their desired name as input, and the game system stores this name for identification and scoring purposes.
Actors: Player
Preconditions: The game is in the registration phase.
Postconditions: The player's name is registered and associated with their gameplay session.
Flow of Events:

1. The player launches the game and enters the registration phase.
2. The game system prompts the player to enter their name.
3. The player enters their desired name.
4. The game system validates the name and registers it.
5. The game system confirms the successful registration to the player.

4.2.2. Use Case: Ready (includes Game Start and Initialize)

Description: This use case represents the player's readiness to start the game. By indicating their readiness, the player signals their intention to participate in the game.
Actors: Player
Preconditions: The game is in the waiting phase, and the player's name is registered.
Postconditions: The game starts and initializes the game session.
Flow of Events:

1. The player launches the game and enters the waiting phase.
2. The game system displays a "Ready" button or prompt.
3. The player clicks the "Ready" button or indicates their readiness.
4. The game system checks if all players have indicated readiness.
5. If all players are ready, the game system starts and initializes the game session.
6. The game system proceeds to the gameplay phase.

4.2.3. Use Case: Word Guess (includes Submit Guess)

Description: This use case allows the player to make a guess by submitting a character or a complete word.
Actors: Player
Preconditions: The game is in the gameplay phase, and it is the player's turn.
Postconditions: The player's guess is evaluated, and the game progresses accordingly.
Flow of Events:

1. The player's turn begins.
2. The game system displays the partially revealed word and any relevant information.
3. The player enters their guess (a character or a complete word).
4. The player submits their guess.
5. The game system evaluates the guess.
6. If the guess is correct, the game system updates the word display, awards points to the player, and provides appropriate feedback.
7. If the guess is incorrect, the game system provides feedback and proceeds to the next player's turn.
8. The game system checks if the word has been fully revealed.
9. If the word is fully revealed, the game ends, and the final scores are calculated.

4.2.4. Use Case: Exit

Description: This use case allows the player to exit the game.
Actors: Player

Preconditions: The game is in progress or has ended.
Postconditions: The player exits the game.
Flow of Events:

1. The player decides to exit the game.
2. The player triggers the exit action, either by typing “quit” in client.py or closing the web browser tab.
3. The game system confirms the player's intention to exit.
4. The game system terminates the game session and closes the game interface.
5. The player is returned to the main menu or exits the game application.

Chapter 5: Class Diagram

Class diagram helps the developers keep track of the whole system while developing the application. Furthermore, the class diagram works as a checklist to ensure that all the functional requirements are developed and included in the application.

Diagram: Class Diagram

5.1. Introduction

The class diagram is a visual representation of the classes, their attributes, methods, and relationships within a system. It provides an overview of the structure and organization of the codebase. This chapter documents the class diagram for the system, highlighting the key classes and their interactions.

5.2. Class Diagram Overview

The class diagram consists of the following classes and their relationships

5.2.1. Server.py

Attributes

- words: list
- word: str
- unrevealed_word: str
- players: dict
- registration_order: list
- turn: int
- ready_statuses: dict
- WORD_LIST_FILE: str
- SERVER_URL: str

Functions

- load_words()
- select_word()
- generate_unrevealed_word()
- handle_guess()
- get_players_ready_data()
- get_players_data()

5.2.2. Server_SocketIO

Attributes

- app: Flask

Functions

- on(“connect”)
- on(“register”)
- on(“ready”)
- on(“guess”)
- on(“switch_player”)

5.2.3. Game

Attributes

- socket: Object
- countdownInterval: Object

Functions

- registerName()
- updatePlayerName()
- ready()
- updateGameState()
- updateScore()
- updateUnrevealedWord()
- updateDesc()
- updateStartPlayersList()
- updatePlayersList()
- sendGuess()

5.2.4. Game_SocketIO

Attributes

- app: Flask

Functions

- on(“connect”)
- on(“start_timer”)
- on(“update_ready_players”)
- on(“update_turn”)
- on(“win”)
- on(“wrong”)
- on(“right”)

5.2.5. Client

Attributes

- socketIO: Object

Functions

- handle_connect()
- handle_disconnect()

5.2.6 Client_SocketIO

Attributes

- app: Flask

Functions

- on(“connect”)
- on(“disconnect”)

Chapter 6: Sequence Diagrams

The sequence diagrams provide a visual representation of the flow of main functions within the system. These diagrams depict the chronological order of interactions between different components and entities. They help in understanding the step-by-step execution of various operations and the exchange of messages among different actors. By following the sequence diagrams, one can easily grasp the control flow, dependencies, and interactions between different functions. They serve as a valuable tool for developers and stakeholders to analyze, validate, and optimize the system's behavior and performance.

6.1. Player registers name and ready

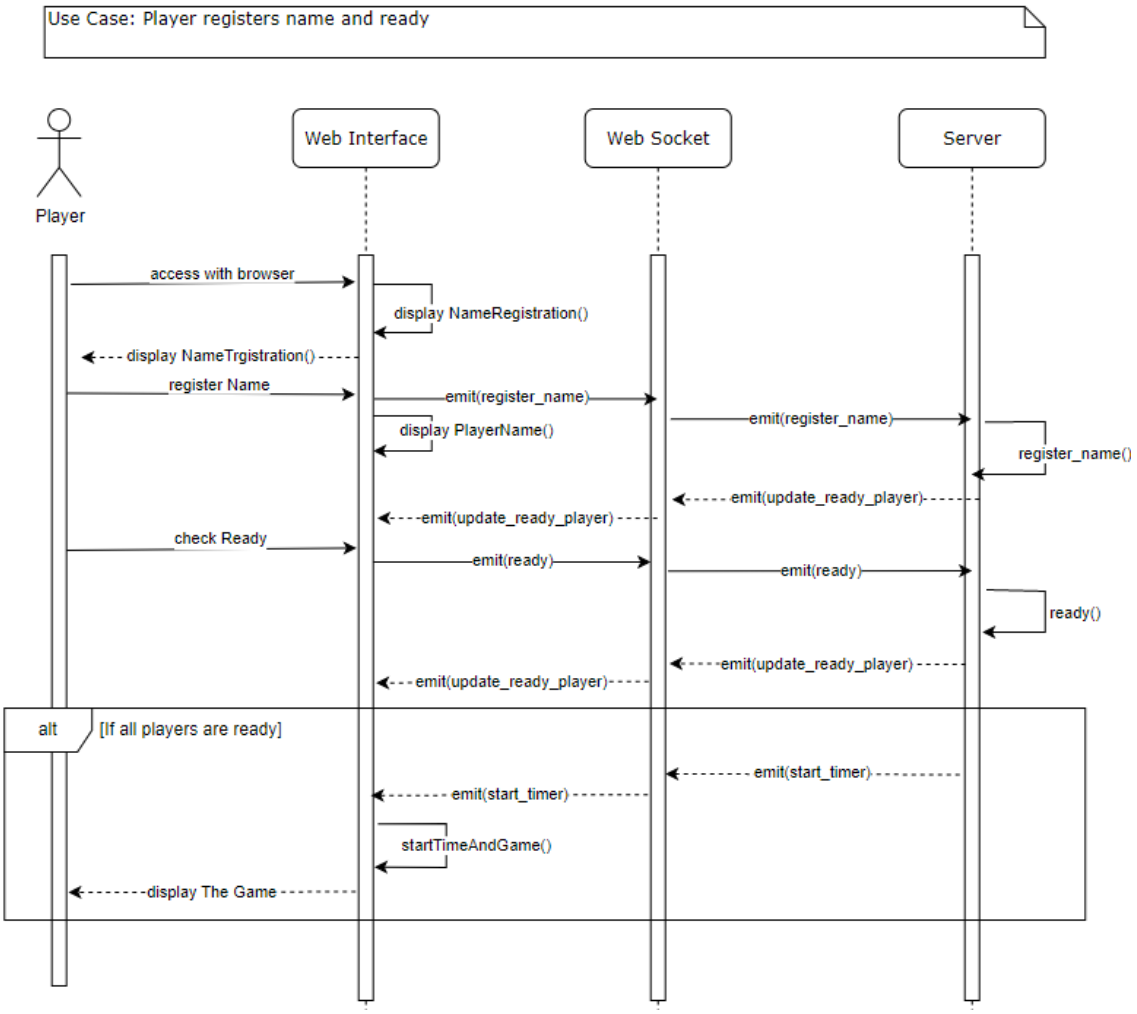

Diagram: [Sequence Diagram]

Use case: Player registers name and ready.
Use case description:

1. Player accesses the web browser.
2. Web browser displays registration and ready checks.
3. Player registers the name.
4. Web browser sends name to server to store in the player list through web socket and displays the name.
5. Player checks ready.
6. Web browser sends ready players list to server through web socket and displays ready status.
7. If all players registered are ready, the game starts and displays to the player.

6.2. Player plays the game

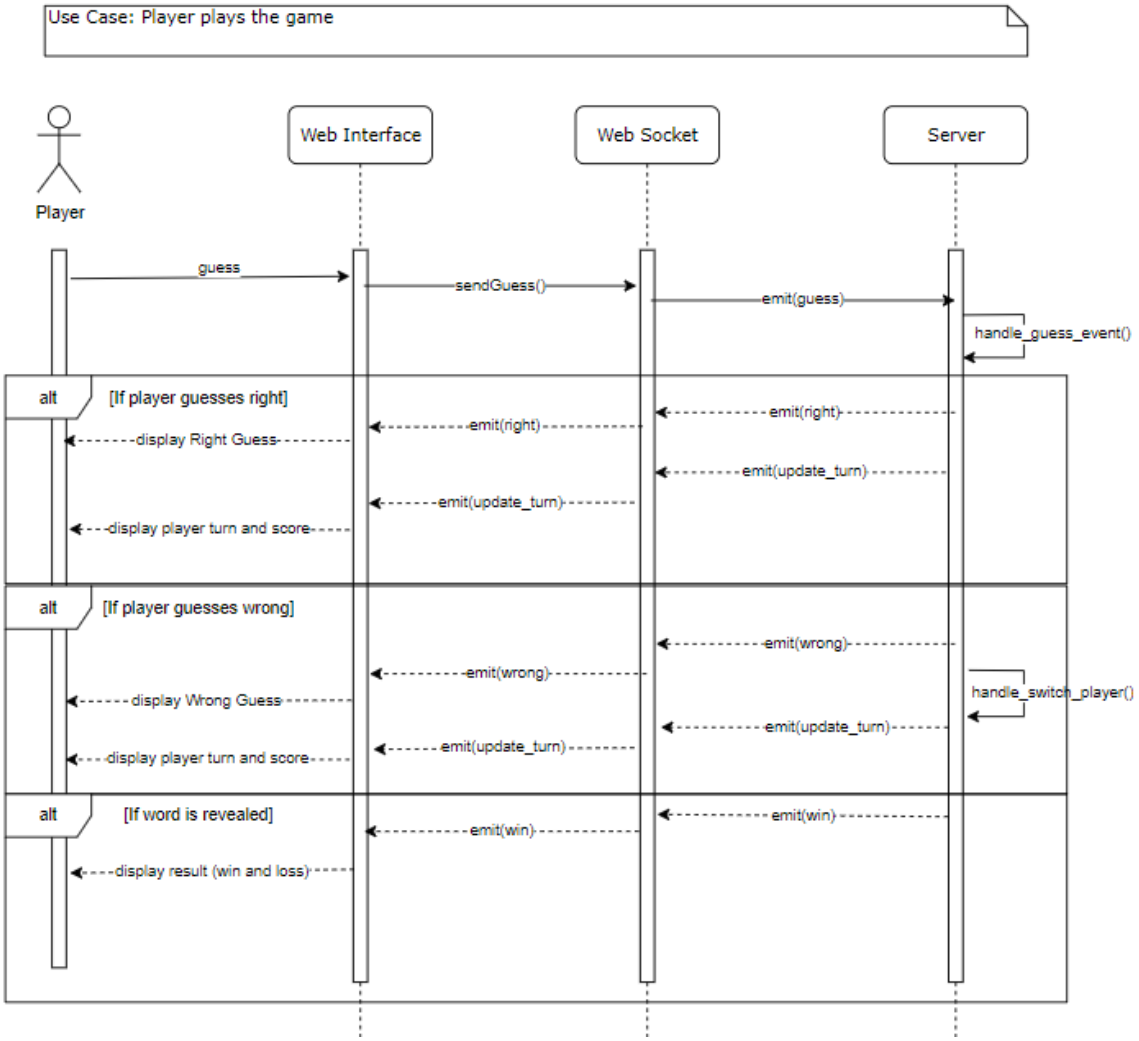

Diagram: Sequence Diagram

Use case: Player plays the game

Use case description:

1. Player guesses the word.
2. Web browser sends the guess to server through web socket.
3. Server handles the guess event.
4. If the guess is correct, server sends win to web browser through web socket. Also, server updates players turn and sends to web browser through web socket.
5. Web browser displays right guess message to player. Then, web browser updates turn and players' scores and the game continues.
6. If the guess is correct, server sends wrong to web browser through web socket. Also, server updates players turn and sends to web browser through web socket.
7. Web browser displays wrong guess message to player. Then, web browser updates turn and players' scores and the game continues.
8. If the word is revealed, server sends win to web browser through web socket.
9. Web browser sends game result to players (win or loss).
10. The game ends.

Chapter 7: Implementation Details

7.1. Programming Languages and Frameworks Used

The implementation of the system involves the use of the following programming languages and frameworks:

1. **Python:** The main programming language used for the backend development of the system. Python provides a wide range of libraries and frameworks that facilitate web development and communication with the client-side.
2. **Flask:** A lightweight web framework for Python that is used for building the backend server. Flask simplifies the process of handling HTTP requests, routing, and rendering dynamic HTML templates.
3. **WebSocket:** WebSocket is a communication protocol that provides full-duplex communication channels over a single TCP connection. It enables real-time, bi-directional communication between the server and the client, making it suitable for interactive applications like the game system.

7.2. Dependencies and Libraries

The implementation of the system relies on various dependencies and libraries to enhance its functionality. Some of the key dependencies and libraries used are as follows:

1. **Flask:** Flask is a web framework for Python that allows you to build web applications. It provides a simple and flexible way to handle HTTP requests, routing, and rendering HTML templates.
2. **json:** The json library provides functions for working with JSON data. In the code, it is used to load words from a JSON file.
3. **random:** The random library provides functions for generating random numbers and making random choices. In the code, it is used to select a random word from the list of loaded words.
4. **Flask-SocketIO:** An extension for Flask that integrates SocketIO functionality into Flask applications. It allows for real-time, event-based communication between the server and the client using WebSockets.
5. **Toastify.js:** A JavaScript library for creating toast notifications. It is used to display various notification messages in the game interface. The code includes a link to the Toastify CSS file (toastify.min.css) and the Toastify JavaScript file (toastify-js).

7.3. File Formats

The system utilizes various file formats for different purposes. The key file formats used are:
JSON (JavaScript Object Notation):

1. **JSON file:** The code loads words from a JSON file. The WORD_LIST_FILE variable specifies the path to the JSON file that contains the word list. The file is expected to be in JSON format, which is a lightweight data interchange format. JSON

(JavaScript Object Notation) is commonly used for storing and transmitting data between a server and a web application. The json library is used to load the JSON data from the file.

2. **HTML template:** The code includes a call to render_template function from Flask, which indicates that there is an HTML template file used for rendering the web page. The template file is typically written in HTML format and may contain placeholders or dynamic content that is filled in by the Flask application.

7.4. Data Structures and Algorithms

The implementation of the system involves the use of different data structures and algorithms to handle various tasks efficiently. Some of the notable data structures and algorithms used are:

7.4.1 Data Structures

1. **Lists:** The code uses lists to store word entries, player names, player scores, and the registration order of players.
2. **Dictionaries:** Dictionaries are used to store player data, including their names and scores. The `game_state.players` dictionary maintains the player information, where the player ID is the key, and the value is a dictionary containing player details.
3. **Strings:** Strings are used to represent words, guesses, and the unrevealed word.
4. **Arrays:** The code utilizes arrays to store player names, player scores, ready statuses, winning players, and losing players. These arrays are used to update and display the game state and player information.

7.4.2. Algorithms

1. **Random Selection:** The `select_word` function randomly selects a word from a list of words.
2. **Word Generation:** The `generate_unrevealed_word` function generates a masked version of a word by replacing characters with dashes (except for spaces).
3. **Guess Handling:** The `handle_guess` function handles the player's guess by comparing it with the word. It updates the unrevealed word based on the guess and returns whether the guess is correct.
4. **Turn Switching:** The `handle_switch_player` function switches the turn to the next player by updating the registration order.
5. **Timer Countdown:** The code includes a timer implemented using JavaScript's setInterval function. It decrements the remaining time for different game events and triggers specific actions when the timer reaches zero.
6. **Updating Game State:** The code updates the game state by modifying various elements on the web page. It uses JavaScript functions to update the unrevealed word, description, player scores, player list, and other game-related information.

7. **Handling Turns:** The code manages player turns by determining the current player based on the turn order and updating the interface accordingly. It includes logic to enable or disable input and button controls based on the current player's turn.
8. **Handling Guesses:** The code handles player guesses by sending the guess to the server through a WebSocket connection. It also includes logic to display toast notifications based on the correctness of the guess and the current player.

Chapter 8: Deployment

To deploy and run the project, follow these steps:

8.1. Install Flask and other dependencies

1. Make sure you have Python installed on your system. You can download Python from the official website (https://www.python.org) and follow the installation instructions.
2. Open a terminal or command prompt and navigate to the project directory.
3. Create a virtual environment (optional but recommended): Run the command `python -m venv venv` to create a virtual environment named "venv".
4. Activate the virtual environment (optional).
5. Install Flask and other dependencies: Run the command `pip install flask flask-socketio`.

8.2. Start the server

1. In the terminal or command prompt, make sure you are in the project directory with the virtual environment activated.
2. Run the command `python server.py` to start the Flask server.

8.3. Access the game via a web browser

1. Open a web browser (Chrome, Firefox, etc.).
2. Enter the following URL in the address bar: `**http://localhost:5000**`.
3. The game should now be accessible, and you can start playing.

Note: If you encounter any errors or port conflicts, make sure there are no other processes running on port 5000. You can change the port number in the `server.py` file if needed.

Chapter 9: Conclusion

9.1. Summary of the project

In this project, we focused on implementing the Python web socket library Flask-SocketIO for the "Wonder Words" game. The goal was to create an interactive and engaging game

within the chatbot's framework. By incorporating Flask-SocketIO, we enabled real-time communication between the chatbot and users, allowing for instant responses and dynamic gameplay. Throughout the project, we successfully developed the "Wonder Words" game, providing users with an enjoyable and interactive experience.

9.2. Challenges faced and lessons learned

During the implementation of Flask-SocketIO for the "Wonder Words" game, we encountered a few challenges. Integrating the web socket library required careful configuration and handling of bidirectional communication. We had to ensure that messages were transmitted accurately and efficiently between the chatbot and users, maintaining a seamless gaming experience.

Additionally, developing the game mechanics and designing engaging word puzzles posed creative and technical challenges. We had to consider various factors such as generating random word puzzles, validating user inputs, tracking game progress, and providing appropriate feedback and scoring mechanisms.

Through these challenges, we learned valuable lessons. Firstly, incorporating Flask-SocketIO expanded the chatbot's capabilities and enhanced user engagement by enabling real-time interactions. It provided a more immersive and interactive experience for users playing the "Wonder Words" game.

Secondly, designing and implementing game mechanics required careful planning and consideration of user experience. Iterative development and user feedback played a crucial role in refining the game's mechanics, ensuring a challenging yet enjoyable gameplay experience.

Lastly, it is a real challenge to ensure that different clients receive synced data on their interfaces. Through the use of Flask socket broadcast functions, the data is transported at the same time to all clients, making the game sync across clients.

9.3. Possible future improvements or extensions

While the implementation of Flask-SocketIO for the "Wonder Words" game was successful, there are several avenues for future improvements and extensions. Here are some possibilities to consider:

1. **Enhanced Chat Functionality:** Expand the chat frame to support additional features such as emojis, file sharing, and voice messaging. This will allow players to express themselves more creatively and engage in richer conversations within the game.
2. **Moderation Tools:** Implement moderation tools to ensure a safe and respectful chat environment. This could include features like profanity filters, automated spam detection, and the ability for players to report inappropriate behavior.

3. **Direct Messaging:** Enable direct messaging functionality so players can privately communicate with their friends and coordinate gameplay or social interactions.
4. **Player Account Enhancements:** Expand the player account system to include customizable profiles, where users can upload profile pictures, write bios, and showcase their in-game achievements. Additionally, consider adding social login options to streamline the account creation process.
5. **Global Chat Channels:** Introduce different chat channels based on topics or interests, allowing players to join and participate in conversations relevant to their preferences. This will encourage community engagement and facilitate interactions among players with similar interests.
6. **Game Room Customization:** Provide players with the ability to create and customize their own game rooms. Allow them to set specific rules, themes, or invite-only access, enabling a more personalized and tailored gaming experience.

These future improvements will enhance the social and multiplayer aspects of the game, fostering a vibrant and connected gaming community. By exploring these future improvements and extensions, the "Wonder Words" game can continue to evolve and offer an even more enjoyable and immersive gaming experience within the chatbot environment.

References

Diagrams link: Diagrams
Flask Socket: Flask-SocketIO — Flask-SocketIO documentation
Net-Cen Requirements: Web Socket Game.pdf
Github: DuyVu285/Wonder-Words-Project (github.com)