SOFTWARE REQUIREMENTS SPECIFICATION

for

Indic Word Search Solver

Prepared by:

Specialization	SAP ID	Name
AIML– B4	500105021	Kanishka Soni
AIML– B4	500106264	Deepanshu Miglani
AIML– B3	500112511	Ayesha Varshney
AIML– B4	500101779	Saikarthik Ayyar

Under the guidance of

Dr. Sahinur Rahman Laskar

UPES
UNIVERSITY OF TOMORROW

School Of Computer Science, University of Petroleum & Energy Studies,
Dehradun- 248007. Uttarakhand

Table of Contents

1. INTRODUCTION

1.1 Purpose of the Project

To develop an Indic word search puzzle game which allows user to play word search puzzle in Indic language including Hindi with the aim of allowing users to play word search solver in their preferred language.

1.2 Target Beneficiary

Those who have strong vocabulary skills in their local language but cannot play word search solvers as many of them are in English hence by incorporating Indic Languages in the game even those whose English is not strong can still enjoy and improve their vocabulary with their preferred Indic Language.

1.3 Project Scope

The project will expand the scope of the word search puzzle game to allow users without fluent English knowledge to learn and improve their vocabulary with the Indic word search puzzle. It benefits in bringing in a large scope of users as it provides an opportunity for both Entertainment and Learning which did not exist previously for those not fluent in English, Its Objective is to allow the user to experience the word search puzzle game in their Indic language with dynamic Interactions to allow maximum accessibility and usability.

1.4 References

[1] Oracle. (n.d.). Swing Documentation. Retrieved from https://docs.oracle.com/javase/tutorial/uiswing/

[2] Introduction to Backtracking https://www.geeksforgeeks.org/introduction-to-backtracking-2/

[3] Dhrubajyoti Pathak, Sukumar Nandi & Priyankoo Sarmah (2024). Evaluating Performance of Pre-trained Word Embeddings on Assamese, a Low-resource Language. Retrieved from https://aclanthology.org/2024.lrec-main.568.pdf

[4] Unicode Table. (n.d.). Unicode Standard for Hindi Documentation. Retrieved from https://www.utf8-chartable.de/unicode-utf8-table.pl?start=2304&number=128

[5] 0xproflupin. (2020). Hindi Dictionary Dataset. Retrieved from https://github.com/0xproflupin/DeepBhashan/blob/master/dataset/Hindi%20dictionary.txt

2. PROJECT DESCRIPTION

2.1 References Algorithm

The algorithm used in this Word Guessing Game project is based on a backtracking approach, utilizing a 2-dimensional array as the primary data structure for the game grid. Backtracking is a recursive method for solving problems by attempting to place a word in various potential grid positions until a valid arrangement is found. In this implementation, each grid cell is checked to see if the word can fit either horizontally or vertically. If it can, the word is placed; if not, the algorithm backtracks to explore other positions. The algorithm checks for boundaries and avoids overwriting previously filled cells, ensuring a feasible and accurate placement. This backtracking approach makes it particularly suitable for a grid-based puzzle, as it enables systematic exploration and guarantees that the word fits without conflicts within the grid structure.

2.2 Characterstic of data

The dataset in this project consists of a collection of Hindi words that will be embedded into a 2-dimensional grid of characters, forming a word puzzle for players to solve. Each word in the dataset is chosen to fit within the grid, which is implemented as a character array where each cell can contain a single letter. The grid is designed to allow words to be hidden horizontally, vertically, or both, depending on the layout defined by the backtracking algorithm. Players can find each hidden word by visually connecting adjacent letters within the grid, with each word's letters arranged in sequential cells, either along a row or a column. This design enables interactive word discovery, where the player must search through various letter arrangements to identify the target word. The careful arrangement of the letters and strategic placement of words ensure that each puzzle remains challenging and unique, requiring players to closely examine letter connections to solve the puzzle accurately.

2.3 SWOT Analysis

Strengths: The software contains randomly Generated grids which allow for dynamic interaction with user, simple usage and freedom of choice to the user that are responded to by the program in real time to allow dynamic interactions with the user.

Weaknesses: Only one word can be present in the crossword and this means the entire grid needs to be randomised again to search for another word hence the software must be restarted to play a fresh puzzle.

Opportunities: Provides multilingual gaming experience that allows multiple people to play the game regardless of their knowledge of English allowing for many users across the world to show their linguistic skills in whatever language that they are most prominent in.

Threats: Relies more on user's observation since it cannot provide the word that is being searched but only confirm its existence as the algorithm requires an input string for tracing the characters in the grid one by one since it is randomly generated.

2.4 Project Features

It has dynamic user interaction that responds to real time choices with difficulty setting features and features in Grid generation, word search and score calculation and to provide accurate hints to the user.

2.5 User Classes and Characteristics

1. **Language Enthusiasts**
    - **Description**: Users who enjoy learning and exploring languages, particularly in their native or preferred Indic languages.
    - **Characteristics**:
    - Strong vocabulary skills in one or more Indic languages.
    - Likely to play the game for entertainment, vocabulary building, or language reinforcement.
2. **Non-English Speakers / Learners**
    - **Description**: Users who aren't fluent in English and thus prefer games in their native language.
    - **Characteristics**:
    - Limited familiarity with English-based word search games.
    - Likely to use the game to improve language skills in their preferred Indic language.
    - May need an intuitive user interface to facilitate smooth gameplay without relying on complex instructions.
3. **Students / Young Learners**
    - **Description**: Students or young learners who are in the early stages of language learning.
    - **Characteristics**:
    - May use the game to learn new words and improve spelling in their language.
    - Benefit from interactive hints and progressive difficulty levels to accommodate learning curves.
    - Enjoy visually engaging features, such as colorful grids and user-friendly navigation.
4. **Casual Gamers**
    - **Description**: Users looking for a casual and relaxing gaming experience in a language they are comfortable with.
    - **Characteristics**:
    - Primarily seek entertainment and leisure without an emphasis on education.
    - May prefer simpler puzzles and a "hint" feature to avoid getting stuck.
    - Enjoy customization options (like themes and difficulty settings) for a personalized experience.

2.6 Design and Implementation Constraints

1. **Multilingual Support**
   The game must support multiple Indic languages with accurate character display using Unicode. A user-friendly interface should allow easy language selection and seamless script handling.
2. **Grid Layout and Word Placement**
   The 2D grid must support dynamic word placement for horizontal and vertical orientations in Indic scripts. The backtracking algorithm should handle alignment and boundary conditions efficiently.
3. **Dynamic Difficulty Levels**
   Difficulty levels should adjust grid size and word complexity, requiring adaptable word placement and scoring mechanisms to ensure balanced gameplay for each level.
4. **Single Word Limitation in Current Grid**
   Currently, only one word can be placed in the grid at a time, requiring a grid reset to start a new puzzle. This limits game continuity and replay without restarting.

2.7 Design Diagrams

2.7.1 Use Case Diagram

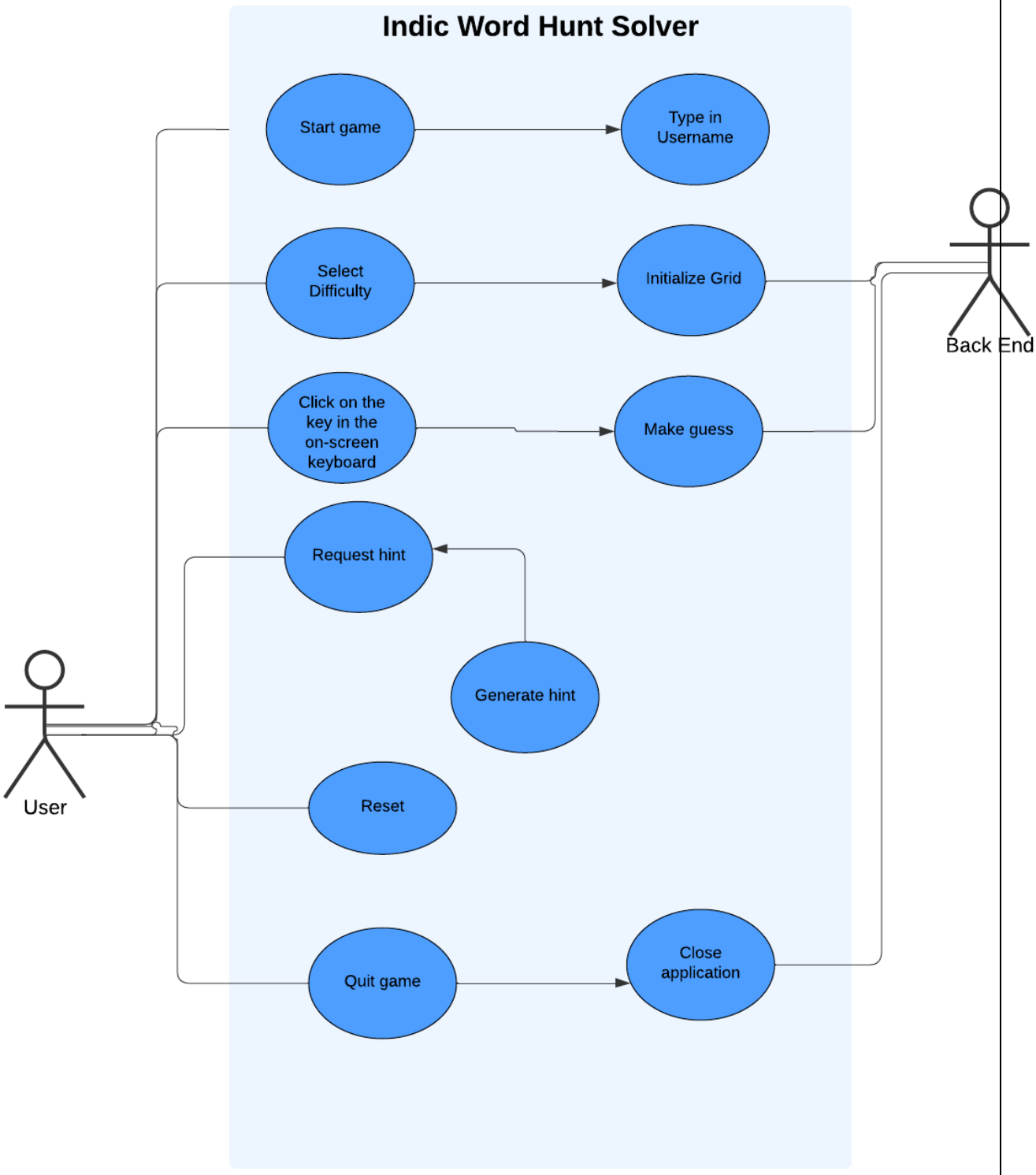

Fig. 2.7.1 Use Case Diagram

2.7.2 Sequence Diagram

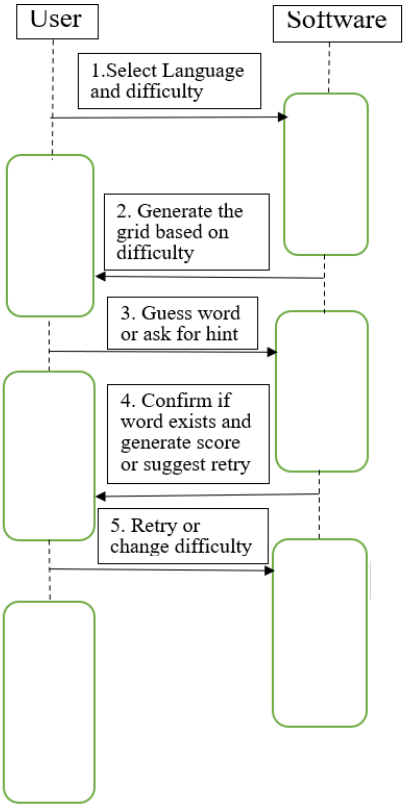

Fig. 2.7.2 Sequence Diagram

2.7.3 Class Diagram

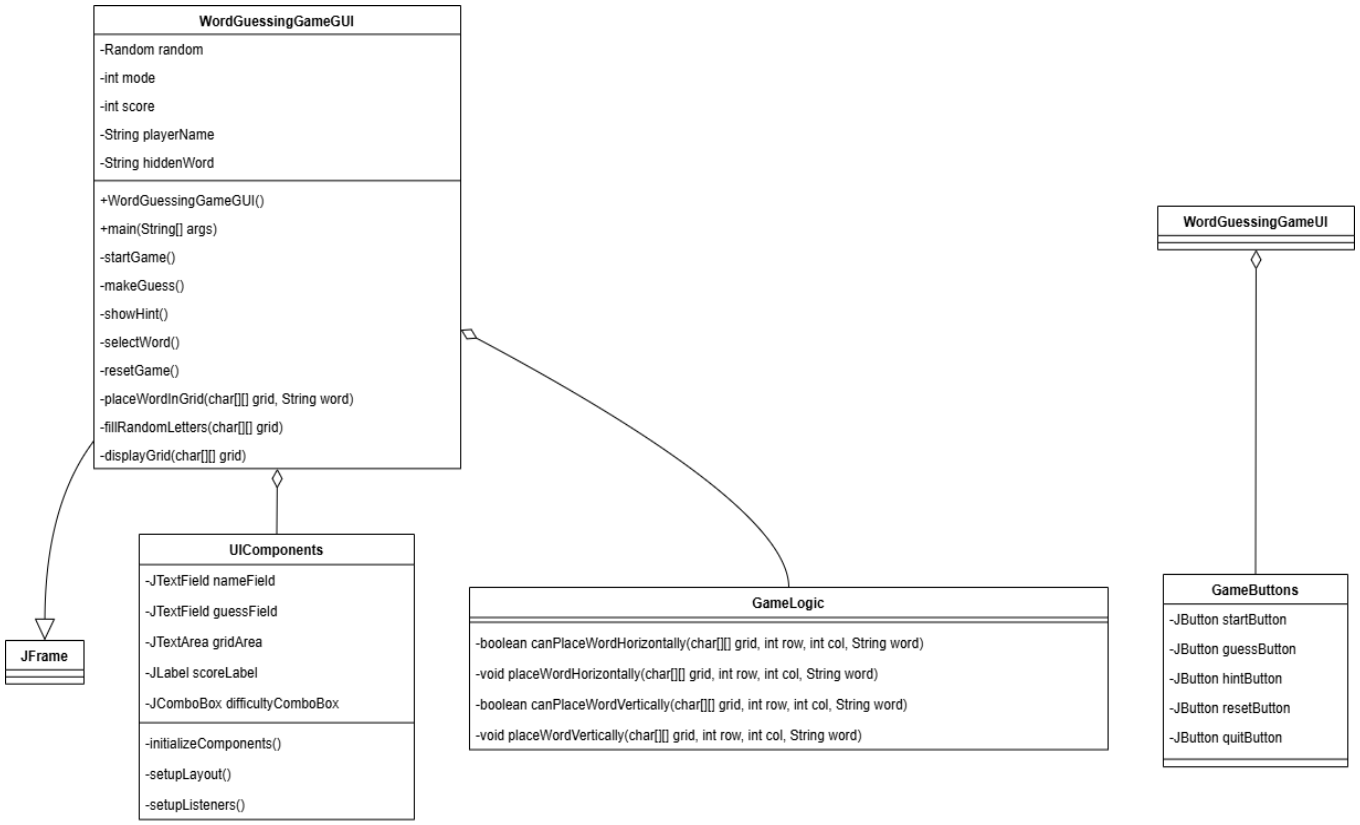

Fig. 2.7.3 Class Diagram

2.7.4 Data Flow Diagram

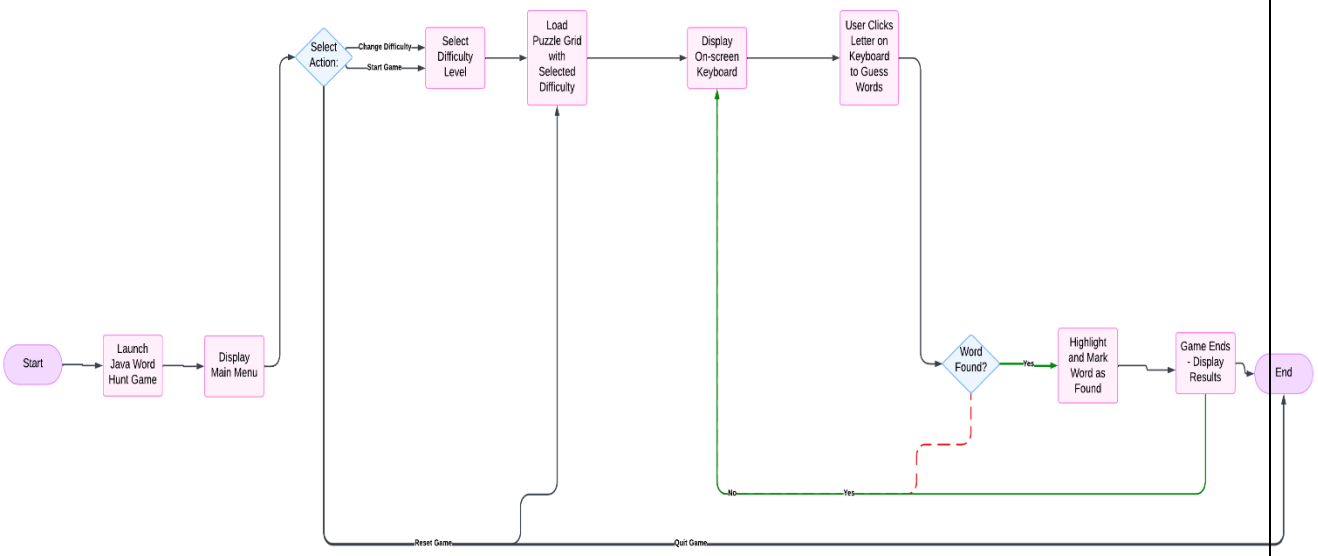

Fig. 2.7.4 Data Flow Diagram

2.8 Assumptions and Dependencies

Assumptions

1. Availability of Indic Language Words
2. User Familiarity with Basic Game Mechanics
3. Unicode Compatibility Across Platforms
4. Single Player Experience

Dependencies:

1. Indic Language Font and Unicode Support
2. Java Swing Library for GUI (or chosen framework)
3. Backtracking Algorithm for Word Placement
4. Access to Word Datasets

3 SYSTEM REQUIREMENTS

3.1 User Interface

The Swing package in Java is imported to create the Graphical User Interface (GUI) for the application. Swing is a part of the Java Foundation Classes (JFC), and it provides a set of GUI components like buttons, labels, text fields, panels, and more, which allow developers to design user-friendly interfaces. By using Swing, the application can provide a more interactive and visually appealing experience for users, allowing them to interact with the program using both the keyboard and the mouse.

3.2 Software Interface

The **Indic Word Search Puzzle Game** is designed to provide an engaging and educational experience for users who prefer to play in their native Indic languages. The game leverages **Java Swing** to create a dynamic and interactive graphical user interface, where users can select their desired language (such as Hindi, Bengali, etc.) and play the puzzle. The word placement is driven by **a backtracking algorithm**, which systematically attempts to place words horizontally or vertically in a 2D grid while ensuring no overlap or conflicts. The game includes features like dynamic difficulty settings, a score tracker, and a language selection option to cater to a wide range of users.

3.3 Database Interface

Currently, the project does not utilize a database. However, integrating a database interface would enhance the game's functionality by allowing the storage and management of user data, game progress, and word lists. A relational database such as MySQL or SQLite can be used to store information like user profiles, scores, and the words available for the puzzles in various Indic languages. This would enable features such as saving user progress, tracking scores over multiple sessions, and offering personalized word suggestions based on language preferences. The database interface would also support dynamic word selection for puzzles and allow users to revisit past games, providing a more interactive and engaging experience.

---

4 NON-FUNCTIONAL REQUIREMENTS

4.1 Performance Requirements

Grid Generation: The backtracking algorithm used to generate word puzzles should be optimized to generate a 2D grid within **2-3 seconds** for grids of moderate size (e.g., 15x15). For larger grids (e.g., 20x20), the system should generate the puzzle within **5 seconds** to avoid any perceptible delays during game startup.

Responsiveness: The game interface should respond to user actions, such as selecting a word or changing difficulty levels, within **1 second**. This ensures a fluid gaming experience without lag or stuttering.

Word Search Efficiency: The word search algorithm should be able to quickly identify and highlight words in the grid, with search times for each word not exceeding **2 seconds**.

Scalability: The system should be capable of handling at least **50 active users** simultaneously without significant performance degradation. This will be important if the game is scaled up to a broader audience.

4.2 Security Requirements

No security requirements due to Abstraction by JAVA ensuring secure data and objects.

4.3 Software Quality Attributes

1. **Usability**: The game interface should be intuitive and easy to navigate, even for users who may not be tech-savvy. Clear instructions, visual feedback, and simple controls should be provided to enhance user experience. The game should also offer multi-language support, particularly for Indic languages, making it accessible to a broader audience.
2. **Performance**: The game should deliver quick response times, with fast puzzle generation and efficient word search algorithms. Performance optimizations should be implemented to ensure that the game remains responsive, even when handling larger grids and complex word searches.
3. **Reliability**: The system should be robust, with minimal crashes or errors. The backtracking algorithm must correctly generate puzzles and place words without failure. In addition, the game should handle edge cases like invalid inputs or unexpected user actions gracefully, providing appropriate error messages when necessary.
4. **Scalability**: The system should be able to handle an increasing number of users without significant performance degradation. This includes managing multiple concurrent game sessions, database queries, and user interactions in an optimized manner.
5. **Maintainability**: The codebase should be modular and well-documented, making it easier to maintain and extend. Future updates, such as adding new languages or features, should be straightforward to implement without introducing major changes to the existing structure.

APPENDIX A: GLOSSARY

Backtracking: A recursive algorithmic technique for incrementally building a solution and backtracking when an invalid state is reached.

Word Solver: A system that helps identify and verify words within a grid of letters.

Indic Languages: A group of languages spoken in the Indian subcontinent, such as Hindi, Bengali, and Tamil.

Graphical User Interface (GUI): A visual interface that allows users to interact with software through graphical elements like buttons and icons.

2D Grid: A two-dimensional array used to represent the game's puzzle board where words are hidden.

APPENDIX B: ANALYSIS MODEL

1. User Input and Interaction: The player interacts with the game via a GUI, selecting options like difficulty and language, and receives feedback such as highlighted words.
2. Grid Generation (Backtracking Algorithm): The system uses a backtracking algorithm to place words into the grid, ensuring no conflicts occur.
3. Word Search Algorithm: The system checks if the selected word exists in the grid and provides feedback by highlighting the word or confirming its presence.
4. Score Calculation: The game calculates the player's score based on words found and time taken, and displays the result.

APPENDIX C: ISSUES LIST

1. Limited Word Grid Generation: Currently, only one word can be placed at a time, leading to the need to restart the puzzle to generate new words, limiting the gameplay experience.
2. Word Placement Conflicts: The backtracking algorithm may fail to find a valid placement for all words, leading to empty grid cells or word overlaps if not handled properly.
3. Performance with Larger Grids: As the grid size increases or the number of words grows, the backtracking algorithm might become slower, potentially affecting game performance.
4. Language Support: While the game supports Indic Languages, the lack of comprehensive support for all regional variations or proper rendering of certain scripts may cause issues for some users.