Word Search Generator

Subject code & section: CSE115.2
Project Group 10
Names and IDs
Nakib Uddin Azadi, Fariba Ahmed, Tawsif Ayon, Safwat Sadiq, Taufiq Omar
2231171043,2512627642,2513269642,2513454642,2512194643

***Abstract* --** The purpose of the project is to develop a Word Search Generator, a program that creates word search puzzles using a given word list. The program places words randomly in a fixed grid horizontally, vertically, and diagonally, filling empty spaces with random letters. The project uses efficient algorithms and data structures like arrays to ensure smooth performance. Users can customize the puzzle by selecting the grid size and difficulty level. Testing evaluates speed, memory usage, and accuracy. With applications in education and entertainment, this tool aids learning while offering fun challenges. Future improvements include a menu system, timer, and different difficulty levels to make the game more interactive and enjoyable**.**

Key words: word search puzzle, word generator, C programming, word grid, word placement strategy.

I. INTRODUCTION

The objective of this project is to develop a Word Search Generator, a program that creates word search puzzles from a given list of words. Word search puzzles help improve pattern recognition and vocabulary skills while providing an engaging experience. The program generates puzzles within a fixed grid, supporting word placement in three orientations: horizontally, vertically, and diagonally. Future improvements include adding a menu with Play and Exit options and expanding the word pool to randomly select 10 words for each puzzle.

II. METHODOLOGY

This word search program involves a systematic approach to grid generation, word placement and direction handling to ensure a challenging yet solvable puzzle.

A. Algorithm Design

The word search generator has a simple and effective algorithm to place words randomly in a 20x20 grid while making sure they don’t overlap. It includes randomized word placement; different directions handling and solve conflicts when words can't fit**.**

B. Grid Generation

The grid is created as a (20x20) 2D array filled with dot characters ('.'). This makes it easy to see and manage where words can go and also easy to

fill the grid with random characters when words are placed**.**

C. Word Placement Strategy

1. Randomized Word Placement: Each word from the words list is placed at a random position in the grid. A random number generator picks the row number and column number where each word starts and an effective algorithm checks for conflicts.

2. Different Direction Handling: Words can be placed in three different directions

- o Horizontally (left to right)
- o Vertically (top to bottom)
- o Diagonally (top to left)

The direction of each word is selected by a random number generator to get randomized direction across the grid.

3. Conflict Resolution: Before placing a word on a certain position in the grid; the program checks if it fits:

- o It makes sure the word does not go outside the grid.
- o It checks if the space is free.
- o It checks if it overlaps with another word.

If there's a problem, it picks a new position and direction until it finds a suitable spot.

4. **Finalizing the Grid**: After all words are placed, all the dot characters ('.') are filled with random capital letters to complete puzzle and make the puzzle look challenging.

This approach ensures that words are placed fairly, the grid looks neat, and players get a balanced, enjoyable and new puzzle every time they play it.

III. IMPLEMENTATION

The word search generator is created completely in the C programming language. The grid is created from a 2D character array and filled with (".") characters. A list of predefined words is used and each word is placed in a random direction and location using randomizer algorithm with use of some standard C library functions. The program checks for available space and only places words if there is no conflict with previously placed ones.

After placing all the words, remaining (".") characters are filled with random uppercase alphabets to complete the grid. The interface includes a simple ASCII art and users are prompted to guess words interactively. When a word is guessed correctly the program removes that word from the grid. It also updates the score afterwards.

IV. FLOWCHART

This flowchart describes the logic of word search generator - from generating the word grid to checking words. Breakdown of the main parts are given below --

A. Start and setup –

- Start the program
- Call printAscii()
- Check if Ascii.txt file exist
  - o if No - print an error message
  - o if Yes -

*Print ASCII art.

*Call gridMaker() to generate a grid

* Call placeWords() to prepare word list and place them.

B. Word placement –

• while (isPlaced == 0) it is initialized as 0 which means no word has been placed yet.

o picks a direction --

* Direction == 0 place words Horizontally

* Direction == 1 place words Diagonally

* Direction == 2 place words Vertically

o (isPlaced ==1) means placement is done successful

• After all words are placed -

o Call randomCharacterplacement () to fill remaining grid spots with random words.

o Call gridPrinter () to display the grid.

C. User interaction –

• word_found_count ==0 , it is initialized as 0 which means no word has been found yet.

• while (word_found_count != 10) all 10 words are not found

o prompt user for input

o Capitalize input

• if word matches and not found yet

o call removeWord() to marks as found.

o (found ==1)found is initialized as 1 which means a word is found.

o if found --

* Print "word found"

* Increment word_found_count()

o else --

*Print "Word Not Found "

* Call printAscii() and gridPrinter () again

D. End Condition –

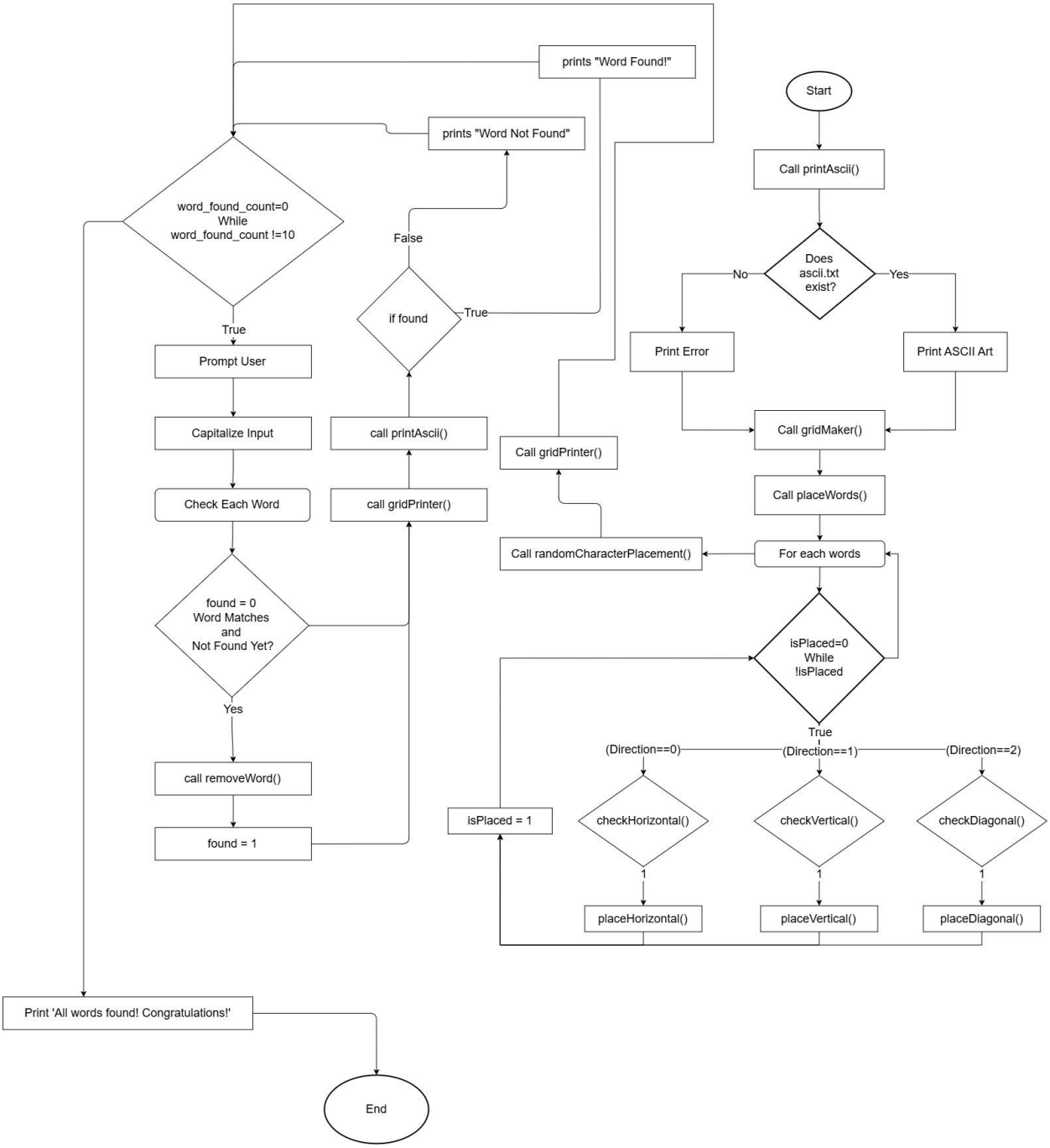
• All 10 words found

o print " All words Found"

o print " Congratulations"

o End

The Flowchart is given below --



V. CHALLANGES

Many challenges were faced during the implementation:

A. Word Overlapping: To check that words do not overlap or goes beyond the grid boundary was a problem. Which needed checking before placement of each word. This was solved by checking direction boundaries and checking each letter's position on the grid.

B. Random Placement Efficiency: Sometimes, mostly for longer words or at the end of the placement process it was hard to find suitable empty spots for the word placement. To solve that, a retry algorithm was implemented to repeat placement attempt until placement is successful.

C. Direction Handling: Generating words in multiple directions required separate logic for horizontal, vertical, and diagonal placements. Separate functions were created to keep the code clean and manageable.

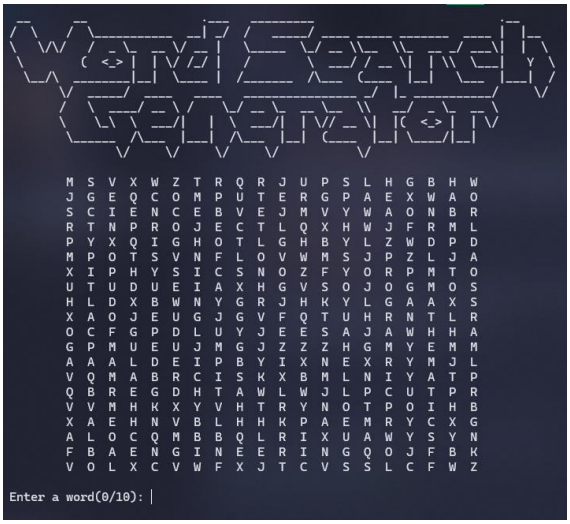
D. Interactive Gameplay: Removing words after the user finds them and updating the grid by removing the guessed words and generating ASCII art and the new grid after that. Additional effort was put into creating a smooth user experience with proper validation and feedback.

Still with all these challenges, the program was successfully implemented and offers an engaging word search experience with dynamic grid generation and interactive play.

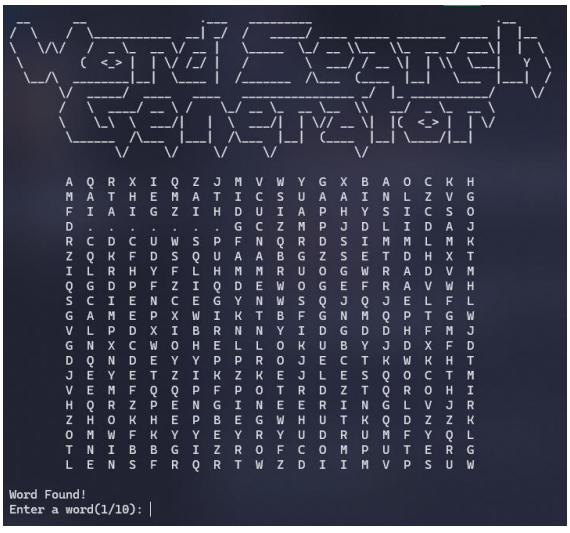
VI. TESTING & VALIDATION

The program was carefully tested using different word lists to make sure it worked properly. It was checked to see if words were placed correctly in horizontal, vertical, and diagonal directions. Special cases, like when words overlap or the grid is too small, were also tested. Every puzzle was reviewed to ensure all the words were included and could be found, showing the program is reliable and accurate.

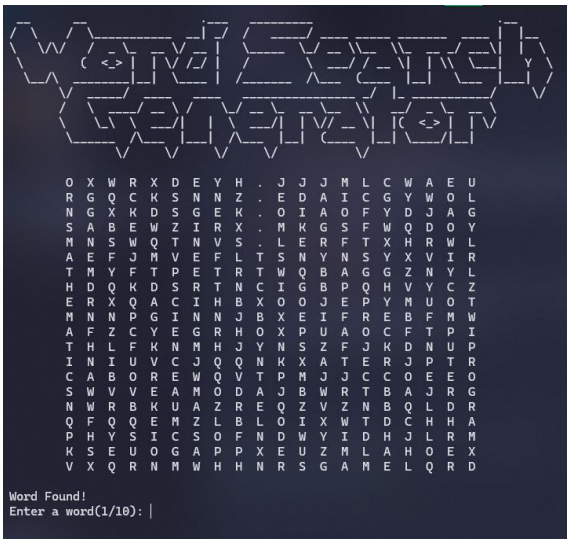
A. Test Cases & Scenarios: Words are placed correctly without any overlapping. The word grid is a 20x20 size without overflow.



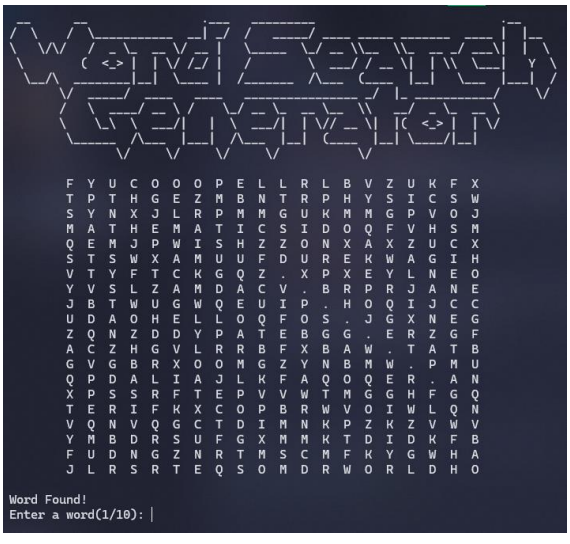
1. Horizontal word placing: Words are placed horizontally (left to right) within grid. Once the word is found it is replaced with (.) character.



2.Vertical word placing: Words are placed vertically (top to bottom) within the grid. Once the word is found it is replaced with (.) character.



3.Diagonal word placing: Words are placed diagonally (top to left) with in the grid. Once the word is found it is replaced with (.) character.



VII. RESULTS

A. Efficiency: Currently the program can correctly generate a 20×20 grid. It can place words diagonally horizontally and vertically. Additionally, it ensures that words are placed without overflow or overlapping issues

***B. Scalability*:** The scalability of this program depends on several factors. Small grids are easy to build and solve but a large grid will increase

complexity of words. Typically, this program can handle a small size grid but for a large grid optimized algorithm might be required**.**

***C. Limitations*:** Right now, the program can only generate 20×20 grid. It places the words, which are included in the program. So, it is unable to generate different words every time.

VIII.APPLICATIONS

A. Educational: This program can be used as a valuable educational tool. It can help the user to practice spelling and building vocabulary. Additionally, it improves pattern recognition and visual scanning skill while boosting problem solving abilities.

B. Entertainment: It can be enjoyed as an entertainment source offering a relaxing yet simulating brain game**.**

IX. FUTURE WORK

To enhance the project, some additional functions can be added, making it more engaging, intuitive, and fun compared to the current version. Additionally, adding a timer can make the game more challenging.

Different levels can be introduced, allowing the user to choose the grid size (10×10,15×15) or the

complexity of the game. A menu can be added with play and exit opinion to make it more user friendly.

X. CONCLUTION

The Word Search Generator is an effective tool for both education and entertainment. It helps users improve vocabulary, pattern recognition, and problem-solving skills while providing a fun and engaging experience. The program efficiently places words in a 20x20 grid, ensuring a balanced and challenging puzzle each time. By using a well-structured algorithm, it guarantees smooth performance and accuracy in word placement. Testing confirms that the program is fast, memory-efficient, and user-friendly. It has practical applications in schools, language learning, and recreational activities. Future improvements, such as a timer, difficulty levels etc. will enhance the gaming experience further. Overall, this project demonstrates the importance of combining learning with entertainment. With continuous development, the Word Search Generator can become an even more interactive and enjoyable tool for users of all ages.

XI. REFERENCES

For knowledge:

1.https://www.w3schools.

com/c/index.php

was used to understand the

basic syntax of certain C

keywords and functions.

2.https://www.geeksforge

eks.org/c-programming-

language/

was used to gain more knowledge

about arrays and pointers.

3. https://www.learn-

c.org/en/Pointers

to learn better implementation of

pointers in the program

4. https://www.ieee.org

for report formatting

writing correction: ChatGPT