

**PF FINAL PROJECT**

**Project Report**

**WORD PUZZLE**

**MUHAMMAD AHMAD**

**TABLE OF CONTENTS**

**ABSTRACT……….……………………………………………………………………..3**

**INTRODUCTION…....…………………………………………………………………..4**

**METHODOLOGY.…….………………………………………………………………...6**

**FLOWCHART……….………………………………………………………………….13**

**CONCLUSION………………………………………………………………………….14**

**FUTURE WORKS……………………………………………………………………...15**

1. **ABSTRACT**

The creation of a C++ word puzzle game is the main goal of this project. Finding words inside a grid of characters is the object of the game. The main goal is to improve vocabulary and pattern recognition abilities while offering a fun game environment.

The game's game board, where characters are randomly positioned, is represented by a 2-dimensional array. Through a command-line interface, the player communicates with the game. Because the board is originally filled by reading from a file, customization, and variation are possible.

There are three different difficulty settings in the game: easy, intermediate, and severe. Each level increases the difficulty by enlarging the search area and adding harder patterns to find. On the console, the player's score and remaining lives are shown at all times.

The project also has features like displaying high scores and preserving game progress. The player can pause the game and pick it back up later by saving the current state to a file. The top five players are kept track of by keeping track of high scores and sorting them in ascending order.

Overall, this project offers players a fun and instructive word puzzle game that pushes them to expand their vocabulary. The implementation shows how to use a number of programming topics, such as handling files, multidimensional arrays, manipulating strings, and algorithmic problem-solving methods.

1. **INTRODUCTION**

The given code represents a word puzzle game implemented in C++. The game involves finding words on a grid of letters. The objective is to search for words in a given grid in both forward and backward wise of horizontal, vertical and diagonal directions. The program reads the board configuration from a file, displays the board on the console, and allows the player to input words to search for.

The code utilizes several functions to perform different tasks. The `read\_board` function reads the board configuration from a file and initializes the 2D array representing the board. The `print\_board` function displays the current state of the board on the console. The `print\_score` function shows the player's score and the number of lives remaining. The `update\_highrecord` function updates the high records based on the player's score. The `dictionary\_check` function checks if a word exists in the dictionary file.

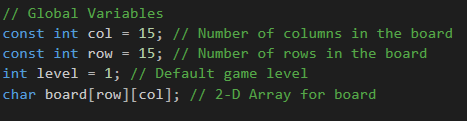
The game offers different levels of difficulty, controlled by the `level` variable. The `easy`, `intermediate`, and `difficult` functions check for word matches in different directions on the board, depending on the level. The player's input is checked against the board and dictionary to determine if it is a valid word.

The game provides additional features such as pausing the game and saving the current state in a file using the `pause\_game` function. The `input\_word` function allows the player to enter words and updates the score and lives accordingly.

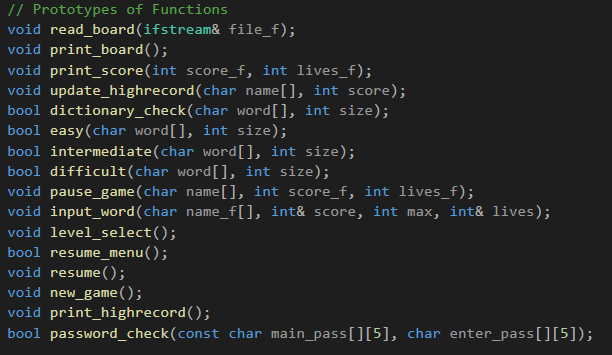
The code also includes a `resume\_menu` function that displays a menu for resuming the game or starting a new one. It prompts the player for input and returns a Boolean value to determine if the game should be resumed or not.

In the following sections of the project report, we will discuss the implementation details of the code, including the data structures, algorithms, and functions used. We will also provide an overview of the game rules, user interface, and the underlying logic behind the word searching mechanism.

1. **METHODOLOGY**

1. **Global Variables**: The code begins by declaring and initializing global variables such as `col`, `row`, `level`, and `board`. These variables represent the dimensions of the board and the current game level.

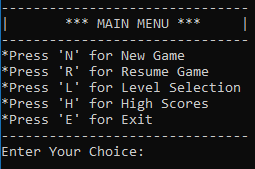
2. **Prototypes of Functions**: The code defines the prototypes of various functions used in the program. These functions are declared before the `main()` function to allow their usage in the program.



3. **Main Function**: The `main()` function serves as the entry point of the program. It displays a welcome message and prompts the user to enter their ID and password. It verifies the entered ID and password using the `password\_check()` function.

4. **Password Check**: The `password\_check()` function compares the entered ID and password with the predefined values stored in the `id` array. If the entered ID and password match, the function returns true; otherwise, it returns false.

5. **Main Menu Loop**: Once the user is successfully authenticated, the program enters the main menu loop. It displays a menu with different options and prompts the user to select an option. The loop continues until the user enters 'e' or 'E' to exit.

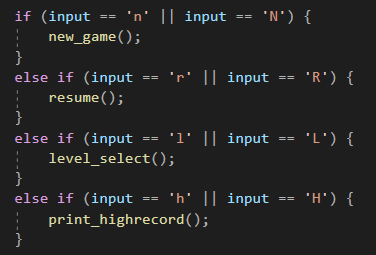


6. **Menu Options**: Depending on the user's choice, the program calls different functions:

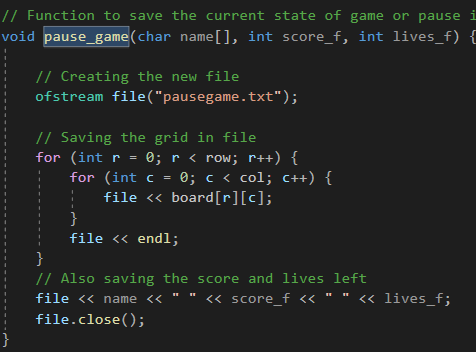
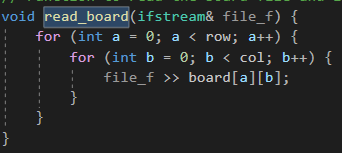
- New Game: If the user selects 'N' or 'n', the `new\_game()` function is called.

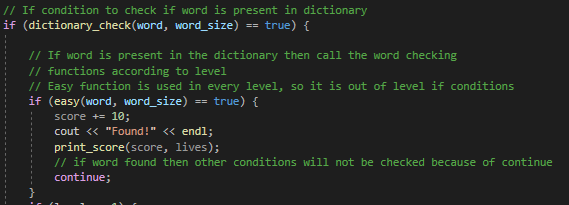
- Resume Game: If the user selects 'R' or 'r', the `resume()` function is called.

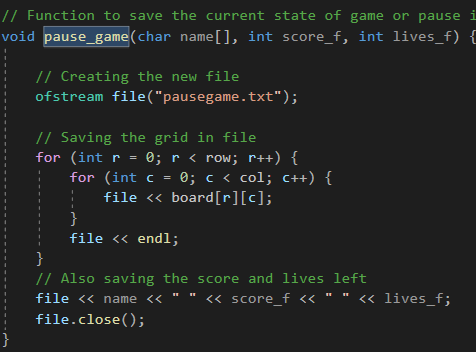
- Level Selection: If the user selects 'L' or 'l', the `level\_select()` function is called.

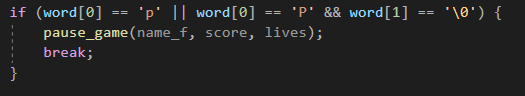
 - High Scores: If the user selects 'H' or 'h', the `print\_highrecord()` function is called.

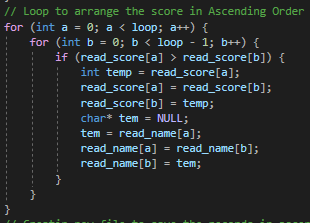
7**. Game Functions**: The code includes various functions that are used during the gameplay. These functions perform tasks such as reading the board from a file, printing the board, printing the score, updating the high records, checking words in the dictionary, checking words in different directions, pausing the game, and taking input from the user.

8. **File Operations**: The code includes file operations to read and write data. It uses file streams (`ifstream` and `ofstream`) to read the board from a file and update the high records.

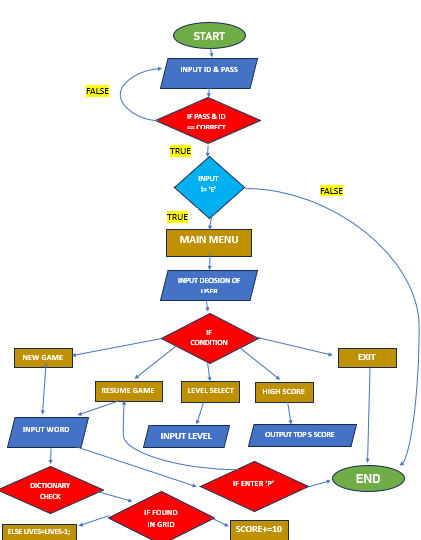
9. **Word Checking**: The functions `dictionary\_check()`, `easy()`, `intermediate()`, and `difficult()` are used to check if a word exists in the dictionary or on the game board. These functions scan the board in different directions (row-wise, column-wise, diagonal) to find the word.

10. **Pause Game**: The `pause\_game()` function saves the current state of the game by writing the board, score, and remaining lives to a file called "pausegame.txt".

11. **Input Word**: The `input\_word()` function allows the user to enter words for the puzzle. It takes input from the user and checks if the word is a valid word on the board. It also handles the case where the user enters 'p' or 'P' to pause the game.

12. **Update High Records:** The `update\_highrecord()` function updates the high records by reading the existing records from a file, comparing the current score with the stored scores, and arranging the records in descending order. It then writes the updated records back to the file.

The methodology describes the overall structure and functionality of the code, explaining how different functions are used to implement the game logic, handle user input, and manage game state.



**5. CONCLUSION**

In conclusion, the code is presented as a Word Puzzle game that lets players solve problems by looking for words in a grid. The game has several different difficulty settings and features like saving and restarting the game state. Different programming concepts and approaches were used throughout the development process to produce a pleasant and useful game experience.

The Word Puzzle game shows how algorithms can be used to search and check words in various ways within the puzzle grid. Since only legitimate words are recognised thanks to the use of a dictionary for word validation, the gameplay is made more difficult and authentic.

Players can save and resume their work thanks to the integration of file handling technologies, which improves the game's usability and convenience. The use of high records demonstrates the capacity to maintain and update data over various game sessions, encouraging players to improve their scores and compete with one another.

Users can easily navigate through menus, choose game options, and interact with the puzzle grid thanks to the user-friendly user interface design of the game. The interface's simplicity allows players to concentrate on the puzzle-solving portion of the game without being distracted by unneeded complexity.

The development of the Word Puzzle game also demonstrates the power and flexibility of programming in developing of interactive and captivating apps. The game encourages the development of critical thinking abilities, vocabulary growth, and general intellectual stimulation by using programming concepts and techniques.

**5. FUTURE WORKS**

Although the Word Puzzle game's current implementation is useful, there are still a number of opportunities for advancement. Future projects could include the following:

1. **Including a graphical user interface (GUI)**: Including a GUI could improve the game's visual appeal overall and give players a more engaging experience. A more interesting gaming experience could be created by adding graphical components like animations and themes.
2. **Adding multiplayer functionality**: Adding multiplayer features would let users work together or compete against one another, promoting friendly competition and social interaction. Implementing local multiplayer options for shared screens or devices or internet multiplayer modes may be required for this.
3. **Increasing the word database**: Adding more words to the game's vocabulary would make the puzzles more varied and challenging. Specialised word lists, themed puzzles, or user-generated content might all be included to create a hard gameplay experience.
4. **Including other game modes**: Adding more game modes, such as timed challenges, word counts that are restricted, or special puzzle limits, would diversity the gameplay. By accommodating varied player preferences and ability levels, these changes would provide a more interesting and personalised experience.
5. **Including social features**: Adding social media tools or online leaderboards would let players share their accomplishments, engage in friendly competition with their friends, and create a sense of community inside the game. This would lengthen the game's lifespan and provide players more incentive.
6. **Improving performance and efficiency**: Improving performance through algorithmic tweaks, code refactoring, or memory management improvements will guarantee responsive and fluid gameplay on a variety of gadgets and platforms. Players would have an improved experience thanks to this, and a larger range of hardware specifications would be supported.

The Word Puzzle game can develop into a more refined, feature-rich, and immersive gaming experience by adopting these future works. These improvements would increase the game's lifespan, replay value, and popularity among a wider range of players.