-----------------------------------------------------------

Word Finder

-----------------------------------------------------------

Assignment – Milestone 2

2800ICT Object Oriented Programming

Bennett taylor

S5095512

Q1 – Word Finder:

Problem Statement:

The goal of this project is to create a program that finds hidden words in a table of characters. The program will work the same as game word search, where hidden words can be found in the table horizontally, vertically, diagonally and, in all cases, in reverse. The program will use the words contained in Dictionary.txt as all possible words that can be found. The program will generate its table of character either by randomly generated lowercase letters or the letters used in puzzle.txt.

User Requirements:

* The user will be able to input the table size(n) when compiling the program and the program will generate a n by n table of random lowercase letters.
* The user will be able to input the word length when compiling, the program will then find all words of that length in the table.
* The user will be able to input a text file when compiling, the program will then load all characters from this text file into the table.
* The user will be able to input the table size followed by the word length when compiling.
* The user will be able to input a text file followed by the word length when compiling.
* The user will only be able to input the table size/ the table size followed by the word length/ a text file/ a text file followed by the word length.
* If the user inputs anything other than above (such as the word length followed by a text file), the program will print:

“I Invalid input, options are:

<game size>

<game size> <word size>

<file name>

<file name> <word size>”

Instructions:

1. Unpack Taylor\_s5095512.zip
2. Open terminal of your choosing
3. Locate file directory in terminal Eg. H:/Uni/OOP
4. Compile the program by typing “g++ Q1.cpp” in terminal. Can also name the executable by typing “g++ Q1.cpp -o Q1” in terminal.
5. Run the compiled program with the table size/text file and the optional word length.

Eg. ./a.out puzzle.txt 5

Documentation:

*Vector<string> read\_file(string file\_name)*

This function reads lines from a text file and returns them as a vector of strings. The file is read one line at a time. When each line is read it is added to the vector of strings.

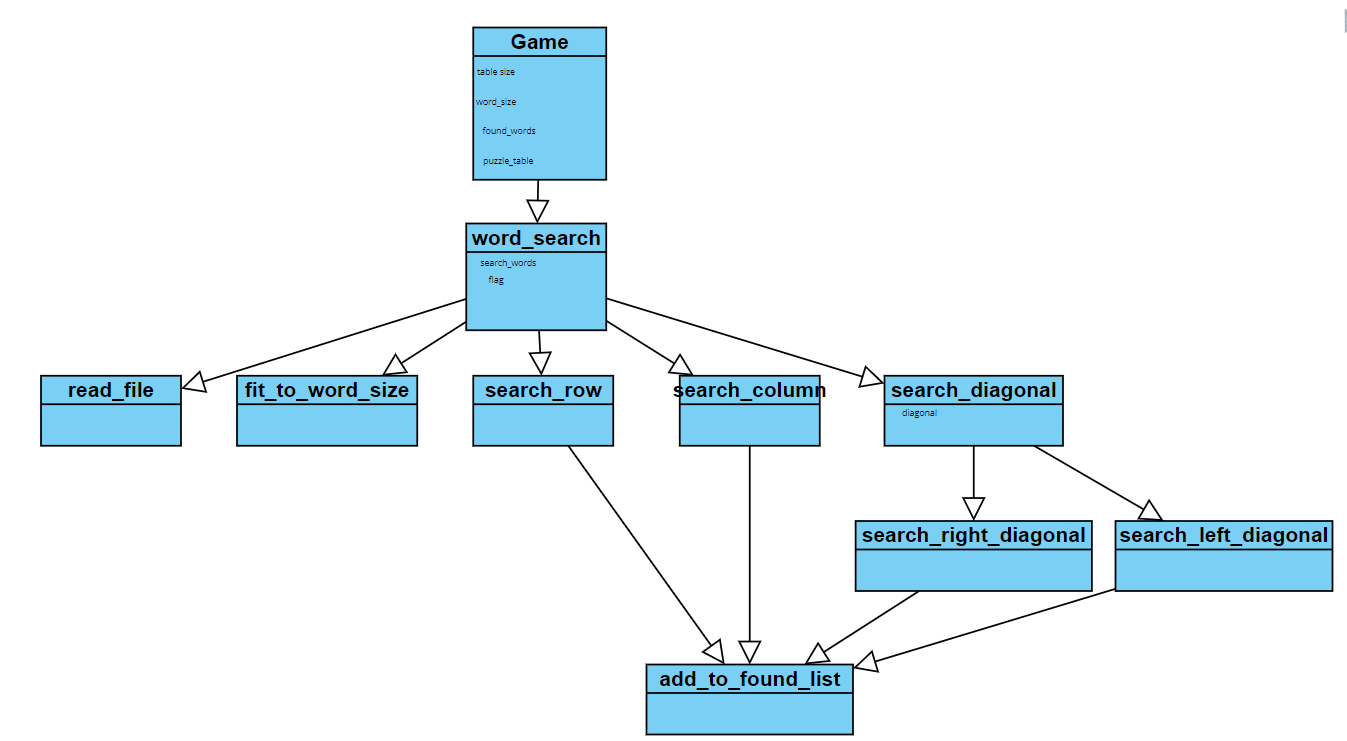
*Vector<string> fit\_to\_word\_size(vector<string> word\_list, int word\_size)*

This function attempts removes all words from a list not equal to word\_size. It does this by creating a new vector of strings, adds all words from a list equal to word\_size then returns this new list.

*Class Game(int argc, char \* argv[])*

This class is responsible for playing the word finder game. It holds all word finder game related functions and stores all word finder related variable. This class is here so the program can make multiple word finder games. Each game can hold different puzzle tables and based on user input. It contains the private variables table\_size, word\_size, found\_words and puzzle\_table.

* Table\_size – defines the size of the puzzle table set by user input
* Word\_size – defines the word length of words to be searched for in the puzzle table
* Found\_words – vector of strings that contains all the words found in the puzzle table. Each string contains the word, the location in the table and description of how the word is read in the puzzle table.
* Puzzle\_table – 2D vector of char’s which holds the values of the puzzle table/grid.

The diagram below displays the flow of the class Game and how each of its member functions are called.

The class starts with its constructor which takes in the arguments argc and argv. Argc contains how many arguments the user has entered when compiling the program and argv is a char pointer that contain those arguments. The constructor first checks argv to see if the user input a number, if so the constructor calls the function create\_random\_table(). If the user did not input a number, it calls the function create\_table\_from\_file(). The constructor then checks if the user has input a word length and sets it to word\_size if they have. The constructor then calls word\_search() to perform the word search on the table that has now been made.

*Game(Game &other)*

This function is a copy constructor part of the class Game. If a Game object is passed in as an argument when initializing a new Game object, this function will override the original constructor. The function first sets all its game variables equal to the game object passed into the function. It then calls word\_search() to perform a word search on these newly set variables.

*Game operator=(Game &other)*

This function is an assignment operator part of the Game class. It returns \*this which is the current object the program is in.

*Std::vector<std::string> found()*

This is a member function of the class Game. It is a getter function that returns the private variable found\_words.

*Vector<vector<char>> get\_puzzle\_table()*

This is a member function of the class Game. It is a getter function that returns the private variable puzzle\_table.

*Int get\_word\_size()*

This is a member function of the class Game. It is a getter function that returns the private variable word\_size.

*Void search\_words()*

This is a member function of the class Game that performs a word search on puzzle table found within the object. It does this by first grabbing the list of words to search for. It calls read\_file() with the argument “Dictionary.txt” to grab a list of all words contained in Dictionary.txt then adds then to the string vector search\_words. The function then checks whether the user has input a word length and removes all words not equal to that length by calling fit\_to\_word\_size(). The enters a nested for loop where the outer layer loops for every word contained in search\_words and the inner layer loops for every row in the puzzle table. Within the two for loops the function calls search\_rows(), search\_column() and search\_diagonals() which search different sections of the puzzle table for the current word in the loop.

*Void search\_diagonals(string search\_word, int row, int flag)*

This is a member function of the class Game that performs a search on both left and right diagonals of the puzzle table. It does this by entering a for loop for each column in the puzzle table. In this for loop functions search\_right\_diagonal() and search\_left\_diagonal() are called.

*Void search\_left\_diagonals(string search\_word, int row, int col, int flag)*

This is a member function of the class Game that performs a search on the diagonals going right to left of the puzzle table. It does this adding each char of a diagonal to a string vector so it’s easier to search. The function then searches the diagonal string to see if the current word is found. If the current word is contained in the diagonal string then the add\_to\_found\_list() is called. The function then reverses the diagonal string to see if the current word is contained but backwards. If the current word is found the function calls add\_to\_found\_list() again.

*Void search\_right\_diagonals(string search\_word, int row, int col, int flag)*

This is a member function of the class Game that performs a search on the diagonals going left to right of the puzzle table. It does this adding each char of a diagonal to a string vector so it’s easier to search. The function then searches the diagonal string to see if the current word is found. If the current word is contained in the diagonal string then the add\_to\_found\_list() is called. The function then reverses the diagonal string to see if the current word is contained but backwards. If the current word is found the function calls add\_to\_found\_list() again.

*Void search\_column(string search\_word, int col)*

This is a member function of the class Game that performs a search on the each column contained in the puzzle table. It does this adding each char of a column to a string vector so its easier to search. The function then searches the column string to see if the current word is found. If the current word is contained in the column string then the add\_to\_found\_list() is called. The function then reverses the column string to see if the current word is contained but backwards. If the current word is found the function calls add\_to\_found\_list() again.

*Void search\_rows(string search\_word, int row)*

This is a member function of the class Game that performs a search on each row contained in the puzzle table. It does this adding each char of a row to a string vector so it’s easier to search. The function then searches the row string to see if the current word is found. If the current word is contained in the row string then the add\_to\_found\_list() is called. The function then reverses the row string to see if the current word is contained but backwards. If the current word is found the function calls add\_to\_found\_list() again.

*Void add\_to\_found\_list(int row, int col, string word, string description)*

This is a member function of the class Game that adds a new word to the found\_words vector.

*Void create\_random\_table()*

This is a member function of the class Game that adds random lowercase letters to the puzzle table.

*Void create\_table\_from\_file(string file\_name)*

This is a member function of the class Game that adds words from a given file to the puzzle table. It does this by taking each individual letter from the file and adding to each space in the puzzle table.

*Void print\_table()*

This is a member function of the class Game and was used for debugging. Its prints the puzzle table to the users screen with added row and column numbers.

*Std::ostream& operator<<(std::ostream &s, Game &game)*

This is an operator over-loader that changes the behavior of << around Game objects. When a Game object its printed using “cout <<” it prints the puzzle table of that Game object.

*bool operator!=(Game &game1, Game &game2)*

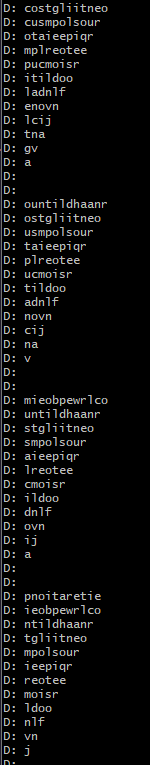
This is an operator over-loader that changes the behavior of != around Game objects. When a Game object is put against another Game object using != this function is called. It behaves the same this normal function (not equal to) but instead compares the variables of each Game object to determine if they are equal or not.

*string operator==(Game &game1, Game &game2)*

This is an operator over-loader that checks if two Game objects are equal.

Testing:

Text

Description automatically generatedDuring testing, a bug was encountered where the program would keep searching the same left and right diagonals. On the right diagonal the program would search starting at (0,0) and go down to the right. The program would then add 1 to column from the for loop contained in word\_search() and start the search at (0,1) and go down the right. This would continue until (0,11), then row is incremented by 1 and column would be set back to 0. The program would then start searching at (1,0) then (1,1) then (1,2)…. This is where the problem was though, the search starting at (1,1) had already been done. It was done when the program searched at (0,0). The same problem was happing to the left diagonal search. To stop this repeating search, a flag was implemented into word\_search() to detect when the program was increasing the row value. The theory was it would not iterate/search/move along the column unless it was at row 0. The diagrams below/to the right display what was happening before and after the flag was implemented. Each line shows the diagonal being searched. Later in the development, it was noticed that the for loop in search\_diagonals() was using puzzle\_table.size() (which = 14) when it should be puzzle\_table[0].size()(which = 12, the correct amount of columns). This could have impacted but due to time constraints this was not investigated further.

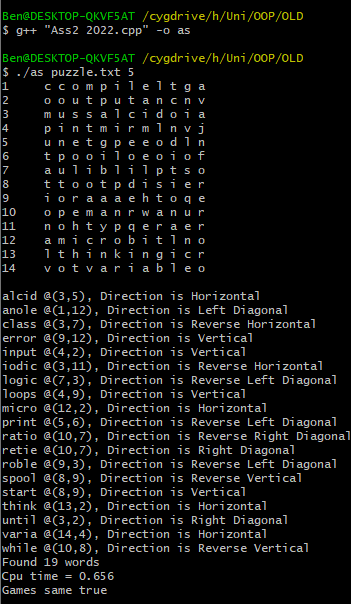
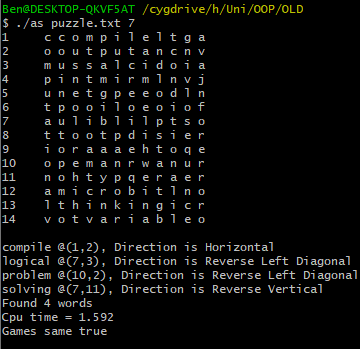
Before

Puzzle Table

A screen shot of a computer

Description automatically generated with low confidence

After

Results: