Word Search

Noah BRAJTMAN

December 2022

Contents

1 Introduction to the subject of the project

The project statement is as follows: we were to create a word search game with predefined classes, namely the Game class, the Board class, the Player class and the Dictionary class.

These classes have some imposed methods, but we will see in this document that not all of them are useful for my game. I therefore took the liberty of taking the subject and modifying it a little. Indeed, I realized a graphic interface using WPF, that's why some of the imposed methods are not relevant for me.

So I created a MainWindow.xaml class which is linked to the main MainWindow.xaml.cs class, in which I create the actions associated to the buttons of my GUI (Game class).

I had no use for the Board class, the Player class and the Game class (which is in fact my MainWindow.xaml.cs). Indeed, managing several classes with WPF is complicated. I however worked on the optimization of my code with the filling of these classes but their handling was too complicated with my interface. You can find the optimization work in the StarUML class diagram.

2 How to play the game

To play the word search game, you must select a difficulty and a language on which two players will compete. They must then fill in their name and click on "Generate" to generate the random grid.

Indeed, a grid will be created according to the parameters entered: the size of the grid and the number of random words selected increase according to the difficulty.

When the grid is generated, the user has a limited time to find the words in the grid, which increases according to the difficulty. The user can select the letters one by one or hold down the left mouse button and drag on the grid. To validate a word, he can either click on the "Verify" button or press the "Enter" key.

At the end of the time limit, a message is displayed announcing the end of the time and showing the score of the first player. It is then the second player's turn. The principle is the same, except that at the end of the time limit, a message announcing the end of the game, the scores and the winner

is displayed.

If a player finds all the words, the game ends instantly and the remaining time will be added to their score. The score is calculated according to the number of letters in each word found, to which is added any time remaining for the player.

3 Details on how the methods used work

The ChangeGrid() method is used to crop the grid according to the difficulty. Indeed, if the difficulty is 1, the grid will be smaller than if the difficulty is 5.

Then, after having put random letters in all the cells of the grid, the AddWordsToGrid() method allows to fill the grid with randomly generated words according to the difficulty. Indeed, the length of the words is greater if the difficulty is greater and the number of words to find is greater if the difficulty is greater. Also, the direction of the words is randomized according to the difficulty. Indeed, for example, for a difficulty of 5, a word can be oriented in a row, in a column or diagonally and forwards or backwards, whereas for a difficulty of 1, the words are only placed in rows or columns forwards.

To check whether a word can be placed in the grid at a certain location, we first check whether the length of the word fits in the grid. Then, if a word is already present, i.e. if the two words intersect, we check that their letters coincide so as not to replace the word placed before. If the conditions are not met, a new location is sought at random.

For this method, I used the RNGCryptoServiceProvider class in order to obtain random values more quickly and with better quality.

Finally, the VerifyWord_Click() method checks whether the word formed by the user belongs to the list of words to be found.