# Word Search Midterm

Midterm C#

Luka Necajev

2020-6-30

ID=991475972

# Table Of Contents

## Quick Overview of everything.

So to start us off I have a Abstract class which will hold the holds all of the necessary abstract and non abstract methods that will be used by the other classes and by the main xaml application to find out if the words are included in the board. Find will call the FindWord() method which will check both the forwards and backwards iterations of the word using the method SearchForwards() this method is inherited by all child classes and is specific to direction and will check to see if you get the complete word. The objects of each class is instantiated in the xaml, and is used to create the board, check user input, make sure that the grid is displayed as well as the lables are updated with the information gotten from the Abstract classes. and everything is working correctly.

# Find.cs

The Find.cs is an abstract class that holds all of the necessary abstract and non abstract methods that will be used by the other classes and by the main xaml application to find out if the words are included in the board. Find will call the FindWord() method which will check both the forwards and backwards iterations of the word using the child classes method SearchForwards() this method is specific to direction and will check to see if you get the complete word.

## FindWord Method

The find word method is used to find the word, it will go through each item in the board and depending on the class that is hosting it will implement the SearchForwards method which will search forwards and backwards in both directions. This method will return the words counted so the program will know how many words are counted.

```
namespace Luka_Necajev_Midterm_WPF
{
    class Find
    {
       //this is just so then I can pass the board through this class.
        static char[,] passBoard;

        public int FindWord(char[] word, char[,] Board, int[,] Board1, int Length, int
numWordsCounted)
        {
            //restarting the number of words counter
            numWordsCounted = 0;
            char checkLetter;
            //function will reverse the word.
            char[] wordReversed = Reverse(word);
            //this will loop through the rows.
            for (int r = 0; r < Length; r++)
            {
                //Debugging
```

```
                Console.WriteLine("r is{0}", r);
                //for each row check all columns
                for (int c = 0; c < Length; c++)
                {

                    int[] possibleIndexX = new int[word.Length];
                    int[] possibleIndexY = new int[word.Length];
                    Console.WriteLine("y is{0}", c);
                    //if it is within the board and has not already been chosen.

                    //DEBUGGING
                   // Console.WriteLine("the board value is{0}", Board[r, c]);
                    //Console.WriteLine(numWordsCounted);
                    checkLetter = Board[r, c];

                    //will check if the letter taken from the board is == to the first
letter of the given word or if it is the last letter of the word.
                    if (checkLetter == word[0])
                    {



                        //creates an array that will store the letters

                        numWordsCounted = searchForwards(r, c, word, Board, Board1,
numWordsCounted, Length);




                    }//end of if letter = word[0]
                    //if the letter is reversed stop loop and add display.
                    if (checkLetter == wordReversed[0])
                    {
                        numWordsCounted = searchForwards(r, c, wordReversed, Board,
Board1, numWordsCounted, Length);

                    }

                    //at a last ditch effort search backwards diagnally.


                }//end of width for loop
            }//end of length for loop
            return numWordsCounted;
        }
```

## SearchForwards Method

It is implemented from the Find Word method, and is an abstract method that will be changed by every class depending on the direction it is looking. This method will search if the first letter is the same as the word, then it will check to see if the next letter is also in the word, in a specific direction that the classes are specified to. This method will return the words counted so the program will know how many words are counted.

```
        //will search for each way depending on the class.
        public virtual int searchForwards(int r, int c, char[] word, char[,] Board,
int[,] Board1, int numWordsCounted, int Length)
        {

            return numWordsCounted;
        }
```

## WordIsCorrect Method

This method will check if the word created in the SearchForward method. It converts the chars to strings and compares them. It will also call the UpperCaseWords that will place the indexes of the word that is correct into a second board, which will be searched at the end so that I can show you where it is being found. This method returns the number of words counted in the char[,] board array.

```
        ///will check if the board is correct
        public static int WordIsCorrect(string check1, string check2, char[] word, int
numWordsCounted, int[,] Board1, int[] possibleIndexX, int[] possibleIndexY)
        {

            //if it makes the word add it to the number counted
            if ((check1 == check2) == true)
            {
                UppercaseWords(possibleIndexX, possibleIndexY, Board1);
                //FindAllCaps(Upperword, Board possibleIndexX, possibleIndexY,))

                numWordsCounted++;

                return numWordsCounted;
            }
            return numWordsCounted;
        }
```

## Reverse Method

The Reverse method will take the word and just return it reversed so I can find it later. This method will just return the char array for the word reversed

```
        //making the reversed word the reversed word
        public char[] Reverse(char[] word)
        {
```

```
            //create a array to hold the reversed word
            char[] wordReversed = new char[word.Length];
            for (int i = 0; i < word.Length; i++)
            {
                //adding the letters in reversed order to the word.
                wordReversed[word.Length - i - 1] = word[i];
            }
            return wordReversed;
        }
```

## CreateNewBoard Method

The CreateNewBoard method will create the board by populating it with random letters from the alphabet(an array of Letters) .This method will just return the 2D char array for the board so we can use it and modify it throughout the program.

```
        public char[,] createNewBoard(char[,] Board, int size)
        {
            //declares alphabet array to create grid with.
            char[] alphabet = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };
            //Creating random object to populate the grid with letters
            //Calls random class to get the letter
            Random rand = new Random();

            //creating the board METHOD
            for (int i = 0; i < size; i++)
            {
                for (int c = 0; c < size; c++)
                {
                    //getting random letters and adding it
                    int randIndex = rand.Next(0, alphabet.Length);
                    //adding it to the board.
                    Board[i, c] = alphabet[randIndex];
                }
            }
            //returning the board.
            return Board;
        }
```

## DisplayBoard Method

The DisplayBoard method will display the board, and will also check if the second board(int[,]) has any items filled, if it does, it will change the color of the background and foreground to show you that this is where the word is. This method is void because it directly edits the elements.

```
        public void DisplayBoard(Grid wordGrid, char[,] Board, int[,]Board1,int size)
        {
            for (int i = 0; i < size; i++)
            {
                wordGrid.ColumnDefinitions.Add(new ColumnDefinition());
                wordGrid.RowDefinitions.Add(new RowDefinition());
```

```
            }
            for (int i = 0; i < wordGrid.RowDefinitions.Count; i++)
            {
                for (int j = 0; j < wordGrid.ColumnDefinitions.Count; j++)
                {
                    Label lb = new Label();
                    //lb.Content = alphabet[rand.Next(0, alphabet.Length - 1)];
                    if(Board1[i, j] == 1)
                    {
                        lb.Content = Board[i, j];
                        lb.SetValue(Grid.RowProperty, i);
                        lb.SetValue(Grid.ColumnProperty, j);
                        lb.Background = new
System.Windows.Media.SolidColorBrush((Color)ColorConverter.ConvertFromString("#666666"));
                        lb.Foreground = new
System.Windows.Media.SolidColorBrush((Color)ColorConverter.ConvertFromString("#45ebed"));
                        lb.Opacity = 0.8;
                    }
                    else
                    {
                        lb.Content = Board[i, j];
                        lb.SetValue(Grid.RowProperty, i);
                        lb.SetValue(Grid.ColumnProperty, j);
                    }


                    wordGrid.Children.Add(lb);
                }

            }
        }
```

## PopulateSecondBoard Method

The PopulateSecondBoard method will populate the second board with 0's. In the WordIsCorrect method it will change them to 1's if the char's match the word. And the DisplayBoard Method will check if they are all 0's and then it will display it This method will be void as the int[] Board1  was declared static.

```
        public void PopulateSecondBoard(int[,] Board1, int size)
        {
            for (int i = 0; i < size; i++)
            {
                for (int c = 0; c < size; c++)
                {
                    //adding it to the board.
                    Board1[i, c] = 0;
                }
            }
        }
```

## SetUpperCase Method

The SetUpperCase method will place the indexes of the board that match the word (if they match) into correct into a second board, which will be searched at the end so that I can display

where it is being found in the xaml. This method is void because it only changes the board object and it is static so it will stay.

```
        public void SetUppercase(int[,] Board1, char[,] Board, int size)
        {
            for (int i = 0; i < size; i++)
            {
                for (int c = 0; c < size; c++)
                {
                    //getting random letters and adding it

                    //adding it to the board.
                    if (Board1[i, c] == 1)
                    {
                        //making the letter upercase. and then adding it back to the char
board in the same spots.
                        char upper = char.ToUpper(Board[i, c]);
                        Board[i, c] = upper;
                    }
                }

            }
        }
```

## SetBoard Method

The SetBoard method will set the board in the abstract method so it can be used in the Make Word Method.

```
        //gets and sets the baord so it can be accessed by all of the buttons.
        public void setBoard (char[,] Board) {
            passBoard=Board;
        }
```

## GetBoard Method

The GetBoard method will get the board in the abstract method so it can be used in the Make Word Method.

```
        public char[,] getBoard()
        {
           return passBoard;
        }

    }

}
```

## MainWindow.xaml

The display of the items and set each style on each item

```
<Window x:Class="Luka_Necajev_Midterm_WPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Luka_Necajev_Midterm_WPF"
        mc:Ignorable="d"
        Title="Luka's WordSearch" Height="500" Width="650">

    <Grid>
        <DockPanel Style="{DynamicResource MainDiv}">
            <StackPanel DockPanel.Dock="Top" Height="40">
                <Label Style="{DynamicResource Title}" x:Name="WordSearch"
Content="Luka's Word Search!" Width="184" Height="36" />
            </StackPanel>
            <Grid Style="{DynamicResource DocSideBottom}" DockPanel.Dock="Bottom"
Height="30">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Label  x:Name="Count" Content="Horizontal Found :"
Style="{DynamicResource Found}"/>
                <Label Grid.Column="1" x:Name="Count1" Content="Vertical Found :0"
Style="{DynamicResource Found}"/>
                <Label Grid.Column="2" x:Name="Count2" Content="Diagonal Left Found :0"
Style="{DynamicResource Found}"/>
                <Label Grid.Column="3" x:Name="Count3" Content="Diagonal Right Found :0"
Style="{DynamicResource Found}"/>
            </Grid>

            <StackPanel DockPanel.Dock="Left" Width="198" Style="{DynamicResource
DocSideLeft}">



                <Label x:Name="___No_Name_" Content="What is the Length of the Board?"
Width="186" Height="30" />
                <TextBox x:Name="tbSize" Text="10" TextWrapping="Wrap" Width="120"
TextChanged="tbSize_TextChanged"/>

                <Label  HorizontalAlignment="Center" x:Name="Label" Content="What is your
Word?" Width="115" Height="30" />
                <TextBox x:Name="tbWord"  HorizontalAlignment="Center" Text="cab"
TextWrapping="Wrap" Width="120" TextChanged="TextBox_TextChanged"/>
                <Button  Style="{DynamicResource btnStyle}" HorizontalAlignment="Center"
x:Name="btnNewGame" Content="New Game" Click="Button_Click_1"/>
                <Button x:Name="SearchWord" Style="{DynamicResource btnStyle}"
HorizontalAlignment="Center" Content="New Word" Click="Button_Click"/>

                <GroupBox Header="Change Color" Margin="5,10">
```

```
                    <StackPanel>
                        <RadioButton x:Name="DarkLayout" Checked="rbLayout1_Clicked"
IsChecked="True">Dark Layout</RadioButton>
                        <RadioButton x:Name="LightLayout"
Checked="rbLayout2_Clicked">Light Layout</RadioButton>

                    </StackPanel>
                </GroupBox>

            </StackPanel>
            <StackPanel DockPanel.Dock="Left" Width="14">

            </StackPanel>
            <StackPanel x:Name="stkBoard" DockPanel.Dock="Right" Width="16">

            </StackPanel>

            <StackPanel Width="566"  HorizontalAlignment="Center"
VerticalAlignment="Center"   Background="#a5c6cc"  Style="{DynamicResource MainWordGrid}"
>



                <Grid x:Name="wordGrid" HorizontalAlignment="Center"
VerticalAlignment="Bottom" Style="{DynamicResource WordGrid}">

                </Grid>

                <Grid x:Name="wordGrid1" HorizontalAlignment="Center"
VerticalAlignment="Bottom" Style="{DynamicResource WordGrid}">

                </Grid>
            </StackPanel>
        </DockPanel>
    </Grid>
</Window>
```

## MainWindow.cs

The backend information when you click each button and start the program.

```
namespace Luka_Necajev_Midterm_WPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    ///

    public partial class MainWindow : Window
    {



        public MainWindow()
        {




            //User prompted for first board size.
            char[,] Board = new char[10, 10];

            int[,] Board1 = new int[10, 10];
            InitializeComponent();
            //creating abst class object
            Find abs = new SearchHorizontal();
            //creating the board
            Board = abs.createNewBoard(Board, 10);
            abs.setBoard(Board);
            abs.DisplayBoard(wordGrid, Board, Board1, 10);



        }

        private void grid_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

            ResourceDictionary skin =
                Application.LoadComponent(new Uri("Dictionary1.xaml", UriKind.Relative))
as ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);

            InitializeComponent();


        }
```

## NewGame Listener

This class is made and will check if the user enters a correct value for the word and size text boxes, it will check using regex if they are correct, if they are not, it will prompt the user with the necessary message saying what the problem is.

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    //creating abst class object
    Find abs = new SearchHorizontal();

    bool checkSize = false;
    bool checkWord = false;
    //CHECKING THE Perameters Passed
    //Checking the Size TextBox
    string input = tbSize.Text;
    int size=0;
    //regex for numbers up to 50
    if (Regex.IsMatch(input, @"^(?:[1-9]|[1-4][0-9]|30)$"))
    {
        //convert it to a int.
        size = Convert.ToInt32(input);
        if (size >= 5 && size <= 30)
        {
            MessageBox.Show("The size is " + size);
            checkWord = true;
        }

        else
        {

            MessageBox.Show("Error!, check and Size and try again");
        }
    }
    else
    {

        MessageBox.Show("Error!, check and Size and try again");
    }

    //CHECKING the Word TextBox
    string wordStr = tbWord.Text;
    if (Regex.IsMatch(wordStr, @"^[a-zA-Z]+$") == false)
    {
        MessageBox.Show("Error!, Something is wrong with the word!");
    }
    else
    {
        checkSize = true;
        wordStr = wordStr.ToLower();
        MessageBox.Show("Word is going through" + wordStr);

    }

    if (checkSize == true && checkWord == true)
```

```
            {

                if (size > 12 && size <20 )
                {
                    Application.Current.MainWindow.Height = 600;


                }else if (size > 20){
                    Application.Current.MainWindow.Height = 900;
                    Application.Current.MainWindow.Width = 900;

                }
                //NEW GAME
                //clear all of the grid elements
                wordGrid.Children.Clear();
                wordGrid.ColumnDefinitions.Clear();
                wordGrid.RowDefinitions.Clear();

                //Creating second board to hold all of the spots of used words


                // MessageBox.Show("size" + size);

                //Creating a 2d board with the Same sized baord
                char[,] Board = new char[size, size];
                //creating a int board to check if it was a thing then Capitalize the
letters
                int[,] Board1 = new int[size, size];


                //adding the string items into the array.
                char[] word = wordStr.ToCharArray();

                //creating the board
                Board = abs.createNewBoard(Board, size);


                //number of words found
                int numWordsFound = 0;

                //number of words counted will be added to this array
                int[] numberOfWords = new int[4];


                //creating abst class object
                Find hor = new SearchHorizontal();
                Find vert = new SearchVertical();
                Find DiagLeft = new SearchDiagonalLeft();
                Find DiagRight = new SearchDiagonalRight();

                //numWordsFound[0] = hor.FindWord(word, Board, size, numWordsFound);
                // bool horizontal=hor.FindWord(word, Board1, size, numWordsFound);

                //setting the board for the other button
                hor.setBoard(Board);

                //method to populate the second board.
                hor.PopulateSecondBoard(Board1, size);
```

```
                //setting the number of each
                numberOfWords[0] = hor.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[0];
                numberOfWords[1] = vert.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[1];
                numberOfWords[2] = DiagRight.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[2];
                numberOfWords[3] = DiagLeft.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[3];

                //showing the count on the screen
                Count.Content = "Horizontal Found :"+numberOfWords[0];
                Count1.Content = "Vertical Found :"+numberOfWords[1];
                Count2.Content = "Diagonal Left Found :"+numberOfWords[2];
                Count3.Content = "Diagonal Right Found :"+numberOfWords[3];

                //sets all of the word locations to uppercase and
                hor.SetUppercase(Board1, Board, size);
                //DONT TOUCH
                //Displayes the board at the end.
                hor.DisplayBoard(wordGrid, Board, Board1, size);



                //DISPLAYING second board
                /* for (int i = 0; i < size; i++)
                {
                    wordGrid.ColumnDefinitions.Add(new ColumnDefinition());
                    wordGrid.RowDefinitions.Add(new RowDefinition());
                }
                for (int i = 0; i < wordGrid.RowDefinitions.Count; i++)
                {
                    for (int j = 0; j < wordGrid.ColumnDefinitions.Count; j++)
                    {
                        Label lb = new Label();
                        //lb.Content = alphabet[rand.Next(0, alphabet.Length - 1)];
                        lb.Content = Board1[i, j];
                        lb.SetValue(Grid.RowProperty, i);
                        lb.SetValue(Grid.ColumnProperty, j);
                        wordGrid.Children.Add(lb);
                    }

                }*/



            }
```

```
}

private void tbSize_TextChanged(object sender, TextChangedEventArgs e)
{



}

private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
{

}
```

## New Word Listener

New word is the button that you will press when you are going to check for another word on your board, you will check for the word and change the word This method is used to check if the word is in the current grid, it is the same as the NewGame method, but it will not create a grid.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    bool checkSize = false;
    bool checkWord = false;
    //CHECKING THE Perameters Passed
    //Checking the Size TextBox
    string input = tbSize.Text;
    int size = 0;
    //regex for numbers up to 50
    if (Regex.IsMatch(input, @"^(?:[1-9]|[1-4][0-9]|30)$"))
    {
        //convert it to a int.
        size = Convert.ToInt32(input);
        if (size >= 5 && size <= 30)
        {
            MessageBox.Show("The size is " + size);
            checkWord = true;
        }
        else
        {

            MessageBox.Show("Error!, check and Size and try again");
        }
    }
    else
    {

        MessageBox.Show("Error!, check and Size and try again");
    }

    //CHECKING the Word TextBox
```

```
            string wordStr = tbWord.Text;
            if (Regex.IsMatch(wordStr, @"^[a-zA-Z]+$") == false)
            {
                MessageBox.Show("Error!, Something is wrong with the word!");
            }
            else
            {
                checkSize = true;
                wordStr = wordStr.ToLower();
                MessageBox.Show("Word is going through" + wordStr);

            }

            if (checkSize == true && checkWord == true)
            {

                if (size > 12 && size < 20)
                {
                    Application.Current.MainWindow.Height = 600;


                }
                else if (size > 20)
                {
                    Application.Current.MainWindow.Height = 900;
                    Application.Current.MainWindow.Width = 900;

                }
                //NEW GAME
                //clear all of the grid elements
                wordGrid.Children.Clear();
                wordGrid.ColumnDefinitions.Clear();
                wordGrid.RowDefinitions.Clear();



                // MessageBox.Show("size" + size);


                //creating a int board to check if it was a thing then Capitalize the
letters
                int[,] Board1 = new int[size, size];


                //adding the string items into the array.
                char[] word = wordStr.ToCharArray();


                //number of words found
                int numWordsFound = 0;

                //number of words counted will be added to this array
                int[] numberOfWords = new int[4];


                //creating abst class object
                Find hor = new SearchHorizontal();
                Find vert = new SearchVertical();
```

```
                Find DiagLeft = new SearchDiagonalLeft();
                Find DiagRight = new SearchDiagonalRight();

                //numWordsFound[0] = hor.FindWord(word, Board, size, numWordsFound);
                // bool horizontal=hor.FindWord(word, Board1, size, numWordsFound);


                //method to populate the second board.
                hor.PopulateSecondBoard(Board1, size);


                char[,] Board= hor.getBoard();


                //setting the number of each
                numberOfWords[0] = hor.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[0];
                numberOfWords[1] = vert.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[1];
                numberOfWords[2] = DiagRight.FindWord(word, Board, Board1, size,
numWordsFound);
                numWordsFound = numberOfWords[2];
                numberOfWords[3] = DiagLeft.FindWord(word, Board, Board1, size,
numWordsFound);
               numWordsFound = numberOfWords[3];

                //showing the count on the screen
                Count.Content = numberOfWords[0];
                Count1.Content = numberOfWords[1];
                Count2.Content = numberOfWords[2];
                Count3.Content = numberOfWords[3];

                //sets all of the word locations to uppercase and
                hor.SetUppercase(Board1, Board, size);




                //DONT TOUCH
                //Displayes the board at the end.
                hor.DisplayBoard(wordGrid, Board, Board1, size);
            }
        }

        private void rbLayout1_Clicked(object sender, RoutedEventArgs e)
        {
```

## Dark Code Skin

Changes the color of the UI based on the radio button that is checked

```
Creating instances of the Skins
```

```
            //creating an instance of resource dictionary and setting it to our varius
resource.
            ResourceDictionary skin =
             Application.LoadComponent(new Uri("DarkCode.xaml", UriKind.Relative)) as
ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);
        }
```

## Light Code Skin

Changes the color of the UI based on the radio button that is checked

```
        private void LIGHT(object sender, RoutedEventArgs e)
        {
            //creating an instance of resource dictionary and setting it to our varius
resource.
            ResourceDictionary skin =
             Application.LoadComponent(new Uri("LightCode.xaml", UriKind.Relative)) as
ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);
        }
    }
}
```

## Search Horizontal

Inherits the Abstract method Find, and overwrites the SearchForward() method which will write the code to check for the horizontal direction. The backwards and forwards is both checked due to the parameters you pass to the Find method.

```
namespace Luka_Necajev_Midterm_WPF
{
    class SearchHorizontal : Find
    {

        public override int searchForwards(int r, int c, char[] word, char[,] Board,
int[,] Board1, int numWordsCounted, int Length)
        {
            //saves the indexes
            int[] possibleIndexX = new int[word.Length];
            int[] possibleIndexY = new int[word.Length];


            //creates an array that will store the letters
            char[] wordFormed = new char[word.Length];
            //if it is within the board and has not already been chosen.
            string check1 = new string(word);
            //this will hold the value of the words formed if the first letter matches
the first letter of your word
            string check2;
            //hold is the y coordinate that you are starting on. since it is only moving
right I will add to the hold instead of y.
            int holdX = r;
            int holdY = c;
            //checks for the letter given the length of the word
            for (int i = 0; i < word.Length; i++)
            {
                //checking if the word can be formed from right to left
                //if the hold is larger then the Width of the board, then do nothing
otherwise check if the word is possible.
                if (holdY >= Length || holdY < 0)
                { }
                else
                {


                    if (word[i] == Board[holdX, holdY])
                    {
                        possibleIndexX[i] = holdX;
                        possibleIndexY[i] = holdY;
                        //new var to stop incrementation of y variable.
                        //the array that will store the letters from the board
                        wordFormed[i] = Board[holdX, holdY];



                        //to check the next letter in the next itteration
                        holdX = holdX + 0;
                        holdY = holdY + 1;
                        //makes the char array a string so I can compare it in one line
                        check2 = new string(wordFormed);
```

```
                        //makes the char array a string so I can compare it in one line
                        check2 = new string(wordFormed);


                        //if it makes the word add it to the number counted
                        numWordsCounted=WordIsCorrect(check1, check2, word,
numWordsCounted, Board1,possibleIndexX,possibleIndexY);


                    }
                    else
                    {
                        //end the function
                        //DEBUGGING
                        Console.WriteLine("Break");
                        break;
                    }

                }//end of else
            }//end of wordLen For loop
            return numWordsCounted ;
        }

        //if each one is there it will populate the array with the capital letter of the
item in hand.



    }
```

## Search Vertical

Child of the Abstract method Find, and overwrites the SearchForward() method which will write the code to check for the vertical direction. The backwards and forwards is both checked due to the parameters you pass to the Find method.

```
namespace Luka_Necajev_Midterm_WPF
{
    class SearchVertical : Find
    {
        public override int searchForwards(int r, int c, char[] word, char[,] Board,
int[,] Board1, int numWordsCounted, int Length)
        {

            //saves the indexes
            int[] possibleIndexX = new int[word.Length];
            int[] possibleIndexY = new int[word.Length];


            //creates an array that will store the letters
            char[] wordFormed = new char[word.Length];
            //if it is within the board and has not already been chosen.
            string check1 = new string(word);
            //this will hold the value of the words formed if the first letter matches
the first letter of your word
            string check2;
```

```
            //hold is the y coordinate that you are starting on. since it is only moving
right I will add to the hold instead of y.
            int holdX = c;
            int holdY = r;
            //checks for the letter given the length of the word
            for (int i = 0; i < word.Length; i++)
            {

                //checking if the word can be formed from right to left
                //if the hold is larger then the Width of the board, then do nothing
otherwise check if the word is possible.
                if (holdX >= Length || holdX < 0)
                { }
                else
                {
                    //Debugging
                    //Console.WriteLine("the letter is {0} at spot {1} in the board",
Board[holdX, holdY], i);
                    if (word[i] == char.ToLower(Board[holdX, holdY]))
                    {
                        possibleIndexX[i] = holdX;
                        possibleIndexY[i] = holdY;
                        //new var to stop incrementation of y variable.
                        //the array that will store the letters from the board
                        wordFormed[i] = char.ToLower(Board[holdX, holdY]);

                        //DEBUGGING
                        //Console.WriteLine("the board value inside loop is {0}",
Board[x, hold]);
                        //to check the next letter in the next itteration
                        holdX = holdX + 1;
                        holdY = holdY + 0;
                        //makes the char array a string so I can compare it in one line
                        check2 = new string(wordFormed);

                        //DEBUGGING
                        Console.WriteLine("{0} ==  {1}", check1, check2);
                        //if it makes the word add it to the number counted
                        numWordsCounted= WordIsCorrect(check1, check2, word,
numWordsCounted, Board1, possibleIndexX, possibleIndexY);

                        /*if ((check1 == check2) == true)
                        {
                            numWordsCounted++;
                            return true;
                        }*/
                    }
                    else
                    {
                        //end the function
                        //DEBUGGING
                        Console.WriteLine("Break");
                        break;
                    }//end of else
                }//end of wordLen For loop
            }//end of if letter = word[0]
            return numWordsCounted;
        }
```

```
    }
```

## Search Diagonal Left

Child of the Abstract method Find, and overwrites the SearchForward() method which will write the code to check for the Diagonal going in the  left direction. The backwards and forwards is both checked due to the parameters you pass to the Find method.

```
class SearchDiagonalLeft : Find
    {

        public override int searchForwards(int r, int c, char[] word, char[,] Board,
int[,] Board1, int numWordsCounted, int Length)
        {

            //saves the indexes
            int[] possibleIndexX = new int[word.Length];
            int[] possibleIndexY = new int[word.Length];


            //creates an array that will store the letters
            char[] wordFormed = new char[word.Length];
            //if it is within the board and has not already been chosen.
            string check1 = new string(word);
            //this will hold the value of the words formed if the first letter matches
the first letter of your word
            string check2;
            //hold is the y coordinate that you are starting on. since it is only moving
right I will add to the hold instead of y.
            int holdX = c;
            int holdY = r;
            //checks for the letter given the length of the word
            for (int i = 0; i < word.Length; i++)
            {

                //checking if the word can be formed from right to left
                //if the hold is larger then the Width of the board, then do nothing
otherwise check if the word is possible.
                if (holdX >= Length || holdX < 0 || holdY >= Length || holdY < 0)
                { }
                else
                {

                    //Debugging
                    //Console.WriteLine("the letter is {0} at spot {1} in the board",
Board[holdX, holdY], i);
                    if (word[i] == char.ToLower(Board[holdY, holdX]))
                    {
                        possibleIndexX[i] = holdY;
                        possibleIndexY[i] = holdX;
                        //new var to stop incrementation of y variable.
                        //the array that will store the letters from the board
                        wordFormed[i] = char.ToLower(Board[holdY, holdX]);

                        //DEBUGGING
                        //Console.WriteLine("the board value inside loop is {0}",
Board[holdX, holdY]);
```

```
                        //to check the next letter in the next itteration adds one to the
x and one to the y
                        holdX--;
                        holdY++;
                        //Console.WriteLine("the letter is {0} at spot {1} in the board",
Board[holdX, holdY], i);
                        //makes the char array a string so I can compare it in one line
                        check2 = new string(wordFormed);

                        //DEBUGGING
                        Console.WriteLine("{0} ==  {1}", check1, check2);
                        /* if ((check1 == check2) == true)
                         {
                              for (int j = 0; j < possibleIndexY.Length; j++)
                         {
                             char upper = Board[possibleIndexX[j], possibleIndexY[j]];
                             //changing the index on the other board.
                             Board[possibleIndexX[j], possibleIndexY[j]] =
char.ToUpper(upper);
                         }
                             numWordsCounted++;
                             return true;
                         }*/
                        numWordsCounted= WordIsCorrect(check1, check2, word,
numWordsCounted, Board1, possibleIndexX, possibleIndexY);

                        //
WordIsCorrect(check1,check2,word,possibleIndexX,possibleIndexY);
                    }
                    else
                    {
                        //end the function
                        //DEBUGGING
                        Console.WriteLine("Break");
                        break;
                    }
                }//end of else
            }//end of wordLen For loop
            return numWordsCounted;
        }
    }
```

## Search Diagonal Right

I Child of the Abstract method Find, and overwrites the SearchForward() method which will the code to check for the Diagonal going in the right direction. The backwards and forwards is both checked due to the parameters you pass to the Find method.

```
class SearchDiagonalRight : Find
    {
        public override int searchForwards(int r, int c, char[] word, char[,] Board,
int[,] Board1, int numWordsCounted, int Length)
        {
            //saves the indexes
            int[] possibleIndexX = new int[word.Length];
```

```
            int[] possibleIndexY = new int[word.Length];


            //creates an array that will store the letters
            char[] wordFormed = new char[word.Length];
            //if it is within the board and has not already been chosen.
            string check1 = new string(word);
            //this will hold the value of the words formed if the first letter matches
the first letter of your word
            string check2;
            //hold is the y coordinate that you are starting on. since it is only moving
right I will add to the hold instead of y.
            int holdX = c;
            int holdY = r;
            //checks for the letter given the length of the word
            for (int i = 0; i < word.Length; i++)
            {
                //checking if the word can be formed from right to left
                //if the hold is larger then the Width of the board, then do nothing
otherwise check if the word is possible.
                if (holdX >= Length || holdX < 0 || holdY >= Length || holdY < 0)
                { }
                else
                {
                    //Debugging
                    //Console.WriteLine("the letter is {0} at spot {1} in the board",
Board[x, y], i);
                    if (word[i] ==char.ToLower( Board[holdX, holdY]))
                    {
                        possibleIndexX[i] = holdX;
                        possibleIndexY[i] = holdY;
                        //new var to stop incrementation of y variable.
                        //the array that will store the letters from the board
                        wordFormed[i] = char.ToLower(Board[holdX, holdY]);

                        //DEBUGGING
                        //Console.WriteLine("the board value inside loop is {0}",
Board[x, hold]);


                        //to check the next letter in the next itteration
                        holdX++;
                        holdY++;

                        //makes the char array a string so I can compare it in one line
                        check2 = new string(wordFormed);

                        //DEBUGGING
                        Console.WriteLine("{0} ==  {1}", check1, check2);
                        /*if ((check1 == check2) == true)
                        {
                            numWordsCounted++;
                            return true;
                        }*/

                        /*if ((check1 == check2) == true)
                        {
                            for (int j = 0; j < possibleIndexY.Length; j++)
                            {
```

```
                                char upper = Board[possibleIndexX[j], possibleIndexY[j]];
                                //changing the index on the other board.
                                Board[possibleIndexX[j], possibleIndexY[j]] =
char.ToUpper(upper);
                            }
                            numWordsCounted++;
                            return true;
                        }*/
                        numWordsCounted= WordIsCorrect(check1, check2, word,
numWordsCounted, Board1, possibleIndexX, possibleIndexY);

                        //
WordIsCorrect(check1,check2,word,possibleIndexX,possibleIndexY);
                    }
                    else
                    {
                        //end the function
                        //DEBUGGING
                        Console.WriteLine("Break");
                        break;
                    }
                }//end of else
            }//end of wordLen For loop
            return numWordsCounted;
        }//end of if letter = word[0]
}
```

## Dark Skin Code

The code for the dark skin

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:Luka_Necajev_Midterm_WPF">


    <Style x:Key="btnStyle">
        <Setter Property="Button.Height" Value="35"/>
        <Setter Property="Button.Foreground" Value="#45ebed"/>
        <Setter Property="Button.Background" Value="#3d3d3d"/>
        <Setter Property="Button.Margin" Value="20"/>
        <Setter Property="Button.Padding" Value="10"/>
    </Style>
    <Style x:Key="DocSideLeft">
        <Setter Property="DockPanel.Background" Value="#666666" />
        <Setter Property="DockPanel.Opacity" Value=".8" />
        <Setter Property="DockPanel.Margin" Value="20"/>
        <Setter Property="Grid.HorizontalAlignment" Value="Center"/>
        <Setter Property="Grid.VerticalAlignment" Value="Center"/>
        <Setter Property="Grid.MinHeight" Value="300"/>

    </Style>

    <Style x:Key="DocSideBottom">
        <Setter Property="DockPanel.Background" Value="#666666" />
        <Setter Property="DockPanel.Opacity" Value=".8" />
        <Setter Property="DockPanel.Margin" Value="20"/>
        <Setter Property="Grid.HorizontalAlignment" Value="Center"/>
        <Setter Property="Grid.VerticalAlignment" Value="Center"/>
```

```
    </Style>

    <Style x:Key="Title">
        <Setter Property="TextBlock.FontSize" Value="20"/>
        <Setter Property="TextBlock.Foreground" Value="#45ebed"/>
    </Style>

    <Style x:Key="Found">
        <Setter Property="TextBlock.FontSize" Value="13"/>
        <Setter Property="TextBlock.Foreground" Value="#45ebed"/>
    </Style>

    <Style x:Key="WordGrid">
        <Setter Property="TextBlock.FontSize" Value="12"/>
        <Setter Property="Grid.HorizontalAlignment" Value="center"/>
        <Setter Property="Grid.HorizontalAlignment" Value="center"/>
        <Setter Property="Grid.ColumnSpan" Value="5"/>

    </Style>

    <Style x:Key="MainGridBackground">

        <Setter Property="Grid.Background" Value="#565656"/>
        <Setter Property="Grid.HorizontalAlignment" Value="center"/>
        <Setter Property="Grid.VerticalAlignment" Value="center"/>
    </Style>


    <Style x:Key="MainDiv">
        <Setter Property="DockPanel.Background" Value="#333333"/>
    </Style>

</ResourceDictionary>
```

## Light Skin Code

The code for the dark skin

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:Luka_Necajev_Midterm_WPF">



    <Style x:Key="btnStyle">
        <Setter Property="Button.Height" Value="35"/>
        <Setter Property="Button.Foreground" Value="#FF3B3F"/>
        <Setter Property="Button.Background" Value="#CAEBF2"/>
        <Setter Property="Button.Margin" Value="20"/>
        <Setter Property="Button.Padding" Value="10"/>
    </Style>

    <Style x:Key="DocSideLeft">
        <Setter Property="DockPanel.Background" Value="#A9A9A9" />
        <Setter Property="DockPanel.Opacity" Value=".8" />
        <Setter Property="DockPanel.Margin" Value="20"/>
```

```
        <Setter Property="Grid.HorizontalAlignment" Value="Center"/>
        <Setter Property="Grid.VerticalAlignment" Value="Center"/>

    </Style>

    <Style x:Key="DocSideBottom">
        <Setter Property="DockPanel.Background" Value="#CAEBF2" />
        <Setter Property="DockPanel.Margin" Value="20"/>
        <Setter Property="Grid.HorizontalAlignment" Value="Center"/>
        <Setter Property="Grid.VerticalAlignment" Value="Center"/>

    </Style>

    <Style x:Key="Title">
        <Setter Property="TextBlock.FontSize" Value="20"/>
        <Setter Property="TextBlock.Foreground" Value="#FF3B3F"/>
    </Style>

    <Style x:Key="Found">
        <Setter Property="TextBlock.FontSize" Value="13"/>
        <Setter Property="TextBlock.Foreground" Value="#FF3B3F"/>
        <Setter Property="TextBlock.Opacity" Value="1"/>

    </Style>

    <Style x:Key="WordGrid">
        <Setter Property="TextBlock.FontSize" Value="12"/>
        <Setter Property="Grid.HorizontalAlignment" Value="center"/>
        <Setter Property="Grid.VerticalAlignment" Value="center"/>
        <Setter Property="Grid.ColumnSpan" Value="5"/>
    </Style>

    <Style x:Key="MainGridBackground">

        <Setter Property="Grid.Background" Value="#565656"/>
        <Setter Property="Grid.HorizontalAlignment" Value="center"/>
        <Setter Property="Grid.VerticalAlignment" Value="center"/>
        <Setter Property="Grid.MinHeight" Value="300"/>
    </Style>

    <Style x:Key="MainDiv">
        <Setter Property="DockPanel.Background" Value="#EFEFEF"/>
    </Style>


</ResourceDictionary>
```