MIDTERM ASSIGNMENT

Jiahao Chen

PROG32356
991530024
08/07/20
Prof: Dario Guiao

Table of Contents

Midterm assignment

Search algorithm:

Box of random letters is designed as 2D array. Letters will be randomly generated and placed inside the array.

```
private char[,] board = new char[len, len];

for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < len; j++)
            {
                // generate an index from 0 - 25
                int nextRand = rand.Next(0, alphabet.Length);
                // assigns the letter at that random index for the alphabet array
                char randomLetter = alphabet[nextRand];
                // places the random letter in the 2D board array
                board[i, j] = randomLetter;
            }
        }
```

Each letter is hard coded into alphabet array:

```
private char[] alphabet = new char[26]
      { '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' };
```

Program will prompt user to enter a string and convert that to a char array:

```
word = str.Text.ToUpper().ToCharArray();
```

similarly, a reverse method will be called to set the string backwards for the backwards search functionality:

```
public char[] reverse(char[] arr)
```

```
{
    // declare new array
    char[] arrReversed = new char[arr.Length];
    for (int i = 0; i < arr.Length; i++)
    {
        // new array assigned original array backwards
        arrReversed[arr.Length - i - 1] = arr[i];
    }
    return arrReversed;
}
```

This program only searches the board in one direction. But two different arrays are used to mimic a backwards search. One array for the correct orientation of the word, another array for the backwards orientation of the word.

search(word, rev.reverse(word), board, len));

The search algorithm will consist of 4 different directions: across, down, diagonal up, diagonal down.

Search across: constant row, increment col by 1 each time

arr[a] == board[startRow, startCol + a]

Search down: constant col, increment row by 1 each time

arr[a] == board[startRow + a, startCol]

Search diagonal up: decrease row by 1, increase col by 1 each time

arr[a] == board[startRow - a, startCol + a]

Search diagonal down: increase row by 1 and col by 1 each time

arr[a] == board[startRow + a, startCol + a]

if letter is found at index position, store it in a temporary array

match[a] = arr[a];

```
Check if temporary array matches the word, if match then entire word is found
if (match.SequenceEqual(arr))
        {
            // if equal then return true;
            return true;
        }
```

Highlight algorithm:

Similar to search algorithm. Create a Boolean 2D array of identical size as board.

private bool[,] highlightedBoard = new bool[len, len];

using same initialize method, set all values in highlightedBoard to false.

highlightedBoard[i, j] = false;

While in the search class, rune the child highlight method if word is found in the board:

```
if (searchWord(i, j, arr, board) || searchWord(i, j, arrRev, board))
                    {
                        // pass and set the same hboard, do not want to instantiate as we
want to retrieve all
                        // valuess in every direction of search.
                        // uses virtual method highLightedBoard in child class
                        hboard = highlightedBoard(i, j, arr, hboard);
                    }
```

Highlight method will be virtual in parent class:

```
    public virtual bool[,] highlightedBoard(int i, int j, char[] arr, bool[,] hboard)
    {
        return hboard;
    }
```

In child class, highlight method is overridden:

```
    public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,]
hboard)
    {
        // loops for the length of array
        for (int a = 0; a < arr.Length; a++)
        {
            // sets the boolean values on bool board to be true where the word was found
            hboard[startRow, startCol+a] = true;
```

```
        }
        return hboard;
    }
```

Highlight method will set the Boolean value in highlight board to be true for the position that it matches the regular board.

Then returns that board and after highlight checkbox is checked, draw a new board to print the highlighted labels at that index:

```
if (highlightedBoard[i, j])
                {
                    lbl.Foreground = Brushes.Red; // change label colour
                }
```

Abstract methods:

Parent Search class is abstract:

abstract class Search() {content}

will include 2 base methods:

public double search(char[] arr, char[] arrRev, char[,] board, int len)

- Returns counter to show how many times word was found in each direction

public bool[,] searchHighlight(char[] arr, char[] arrRev, char[,] board, bool[,] hboard, int len)

- Returns the Boolean array board to set flags for where the word appears on the normal board

Will contain 2 virtual methods:

```
public virtual bool searchWord(int i, int j, char[] arr, char[,] board)
{
    return false;
}

public virtual bool[,] highlightedBoard(int i, int j, char[] arr, bool[,] hboard)
{
    return hboard;
}
```

These methods will be overridden in the child classes. Virtual methods are used because in the main class, the base method Search from the abstract class is called. In the Search method, the child class is called to search in each direction and highlight. Therefore a virtual implementation is used.

public override bool searchWord(int startRow, int startCol, char[] arr, char[,] board)

public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,] hboard)

MainWIndow.xaml class:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Chejiaha_Midterm
{
    /// <summary>
    /// Jiahao Chen
    /// 991530024
    /// PROG32356
    ///
    /// Simple word search application utilizing concepts from OOP.
    /// Displays knowledge of 2D arrays, abstract classes, inheritence,and WPF form user interface design.
    /// User will be asked to enter size of grid. Then click on draw button. Once the board is shown, user can
```

```
    /// enter a search string. If the string is found in the board, then user will be notified. User has options
    /// to highlight the words found, indicating their position on the board. User can also adjust text size via
    /// radio buttons.
    ///
    /// </summary>
    public partial class MainWindow : Window
    {
        // declaring global variables
        private static int len = 100;
        // initialize char and bool 2D array of length
        private char[,] board = new char[len, len];
        private bool[,] highlightedBoard = new bool[len, len];
        // all possible letters to be displayed
        private char[] alphabet = new char[26]
        { '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' };
        private char[] word;
        // regex checks if user has entered only numbers
        Regex regex = new Regex("^[0-9]+$");
        // Object calls
        Reverse rev = new Reverse();
        Search sa = new SearchAcross();
        Search sd = new SearchDown();
        Search sdd = new SearchDiagonalDown();
        Search sdu = new SearchDiagonalUp();
        public MainWindow()
        {
```

```
    InitializeComponent();
}

public char[,] initialize()
{
    // creates new random object
    Random rand = new Random();
    // initializes the boolean board everytime a new char board is created
    highlightedBoard = new bool[len, len];
    // iterates through the board
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            // generate an index from 0 - 25
            int nextRand = rand.Next(0, alphabet.Length);
            // assigns the letter at that random index for the alphabet array
            char randomLetter = alphabet[nextRand];
            // places the random letter in the 2D board array
            board[i, j] = randomLetter;
            // sets all values on boolean board to false
            highlightedBoard[i, j] = false;
        }
    }
    return board;
}

// draws out the board in grid
```

```
public void drawBoard(char[,] board)
{
    // clears the grid after before drawing anything
    wordGrid.Children.Clear();
    wordGrid.ColumnDefinitions.Clear();
    wordGrid.RowDefinitions.Clear();

    for (int i = 0; i < len; i++)
    {
        // initialize column and rows of grid
        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++)
        {
            // create a label for each empty cell and populate with 2D array
            Label lbl = new Label();
            lbl.Content = board[i, j]; // set label content to letter in board array
            lbl.SetValue(Grid.RowProperty, i); // set letter in grid row
            lbl.SetValue(Grid.ColumnProperty, j); // set letter in grid column
            wordGrid.Children.Add(lbl); // dynamically add the label into each cell
        }
    }
}

private void displayBoardBtn(object sender, RoutedEventArgs e)
```

```
{
    // check is regex condition is met (only numbers in textbox)
    if (regex.IsMatch(length.Text))
    {
        // set the length of boards equal to value user entered
        len = Convert.ToInt32(length.Text);
        // initialize and draw
        // resets both char board and bool board
        initialize();
        drawBoard(board);
    }
    else
    {
        // if user did not enter a number, alert user
        MessageBox.Show("Please enter a length");
    }
}

private void SearchBtn(object sender, RoutedEventArgs e)
{
    // checks if string is empty
    if (str.Text == "")
    {
        // alert user to enter a value to search
        MessageBox.Show("cannot be empty");
    }
    else
    {
```

```
            // sets the string entered by user to a char array
            word = str.Text.ToUpper().ToCharArray();
            // changes labels in stackpanel to display the number of matches found in
puzzle
            // using global object calls from earlier
            lblAcross.Content = string.Format("Across: {0}", sa.search(word,
rev.reverse(word), board, len));
            lblDown.Content = string.Format("Down: {0}", sd.search(word,
rev.reverse(word), board, len));
            lblDiagonalDown.Content = string.Format("Diagonal Down: {0}",
sdd.search(word, rev.reverse(word), board, len));
            lblDiagonalUp.Content = string.Format("Diagonal Up: {0}", sdu.search(word,
rev.reverse(word), board, len));
            /*if ((bool)(chkHighlight.IsChecked))
            {
                checkBox(object sender, RoutedEventArgs e);
            }*/
        }
    }

    // redraws the board when highlight is unchecked
    private void unChkHighlight(object sender, RoutedEventArgs e)
    {
        drawBoard(board);
    }

    private void checkBox(object sender, RoutedEventArgs e)
    {
        // check if user has entered a value into length before attempting to highlight
        if (str.Text != "" && length.Text !="")
```

```
        {
            // clears the grid after before drawing anything
            wordGrid.Children.Clear();
            wordGrid.ColumnDefinitions.Clear();
            wordGrid.RowDefinitions.Clear();
            highlightedBoard = new bool[len, len];

            // sets the boolean values on the boolean board where the word(s) were found
            // does not override the existing board each time an object is called, to show all 4 directions.
            // Only overwrites the board if new word is requested by user
            highlightedBoard = sa.searchHighlight(word, rev.reverse(word), board, highlightedBoard, len);
            highlightedBoard = sd.searchHighlight(word, rev.reverse(word), board, highlightedBoard, len);
            highlightedBoard = sdu.searchHighlight(word, rev.reverse(word), board, highlightedBoard, len);
            highlightedBoard = sdd.searchHighlight(word, rev.reverse(word), board, highlightedBoard, len);

            for (int i = 0; i < len; i++)
            {
                // initialize column and rows of grid
                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++)
```

```
            {
                // create a label for each empty cell and populate with 2D array
                Label lbl = new Label();
                lbl.Content = board[i, j]; // set label content to letter in board array
                lbl.SetValue(Grid.RowProperty, i); // set letter in grid row
                lbl.SetValue(Grid.ColumnProperty, j); // set letter in grid column
                // if element in highlightedBoard array is true
                if (highlightedBoard[i, j])
                {
                    lbl.Foreground = Brushes.Red; // change label colour
                }
                wordGrid.Children.Add(lbl); // dynamically add the label into each cell
            }
        }
    }
    else
    {
        MessageBox.Show("Please draw the board and search for word first");
    }
}

// skin handlers
// will change the size and padding of labels in word grid

// smaller label
private void rbLayout1_Clicked(object sender, RoutedEventArgs e)
{
    ResourceDictionary skin =
```

```
            Application.LoadComponent(new Uri("Layout1.xaml", UriKind.Relative)) as
ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);
        }
        // bigger label
        private void rbLayout12_Clicked(object sender, RoutedEventArgs e)
        {
            ResourceDictionary skin =
             Application.LoadComponent(new Uri("Layout2.xaml", UriKind.Relative)) as
ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);
        }
        // reset label
        private void rbLayout3_Clicked(object sender, RoutedEventArgs e)
        {
            ResourceDictionary skin =
             Application.LoadComponent(new Uri("Layout3.xaml", UriKind.Relative)) as
ResourceDictionary;
            Resources.MergedDictionaries.Add(skin);
        }
    }
}
```

Reverse class:

```
using System;
using System.Collections.Generic;
using System.Text;

// reverse the string that user enters.
// to be used in search classes, find the word backwards in board
namespace Chejiaha_Midterm
{
    class Reverse
    {

        // reverses the word that user entered into a new array
        public char[] reverse(char[] arr)
        {
            // declare new array
            char[] arrReversed = new char[arr.Length];
            for (int i = 0; i < arr.Length; i++)
            {
                // new array assigned original array backwards
                arrReversed[arr.Length - i - 1] = arr[i];
            }
            return arrReversed;
        }
    }
}
```

Abstract Class:

```
namespace Chejiaha_Midterm
{
    // Parent/abstract class of search methods
    abstract class Search
    {
        public double counter = 0;
        // search method is used in every child class. Does not get overriden.
        // This method checks if the word is on the board and ensures that the program
        // does not check an outofbounds index before continuing to other search methods.
        public double search(char[] arr, char[] arrRev, char[,] board, int len)
        {
            counter = 0; // initialize the counter for matches found
            for (int i = 0; i < len; i++)
            {
                for (int j = 0; j < len; j++)
                {
                    // checks if first letter is on the board
                    // checks if the word entered by user exceeds the legnth of board
                    if ((board[i, j] == arr[0] || board[i, j] == arrRev[0]) && arr.Length <= len)
                    {
                        // if word is found in normal or reverse order then increment the counter
                        // uses the virtual method searchWord in child class
                        if (searchWord(i, j, arr, board) || searchWord(i, j, arrRev, board))
                        {
                            counter++;
                        }
```

```
            }
        }
    }
    return counter; // always return the count for each orientation of search
}

// similar to the previous search method. This one sets the boolean values in boolean board to be true if found.
// this method does not get overriden and is called by every child class.
public bool[,] searchHighlight(char[] arr, char[] arrRev, char[,] board, bool[,] hboard, int len)
{
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            // checks if first letter is on the board
            // checks if the word entered by user exceeds the legnth of board
            if ((board[i, j] == arr[0] || board[i, j] == arrRev[0]) && arr.Length <= len)
            {
                // if word is found in normal or reverse order then increment set value to true
                if (searchWord(i, j, arr, board) || searchWord(i, j, arrRev, board))
                {
                    // pass and set the same hboard, do not want to instantiate as we want to retrieve all
                    // valuess in every direction of search.
                    // uses virtual method highLightedBoard in child class
                    hboard = highlightedBoard(i, j, arr, hboard);
                }
```

```
            }
        }
    }
    return hboard; // always return the boolean board
}

// Declaring virtual mathods. These will be overriden in child classes
// These methods will be overridden in the child classes. Virtual methods are used because in the main class,
// the base method Search from the abstract class is called.
// In the Search method, the child class is called to search in each direction and highlight.
// Therefore a virtual implementation is used.
public virtual bool searchWord(int i, int j, char[] arr, char[,] board)
{
    return false;
}

public virtual bool[,] highlightedBoard(int i, int j, char[] arr, bool[,] hboard)
{
    return hboard;
}
  }
}
```

Search Across (child class):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Chejiaha_Midterm
{
    class SearchAcross: Search // searchDown inherits Search
    {

        // overrides virtual method from abstract class
        public override bool searchWord(int startRow, int startCol, char[] arr, char[,] board)
        {

            char[] match = new char[arr.Length];
            // iterate through the array to find match
            for (int a = 0; a < arr.Length; a++)
            {
                // starts at position where first letter was found, then iterates across (static row coordinate)
                // to check if the rest of word matches the characters on the board
                if ((arr[a] == board[startRow, startCol + a]))
                {
                    // assign temp array values
                    match[a] = arr[a];


                }
```

```
            }
            // check if the array stored is equal to the word user entered
            if (match.SequenceEqual(arr))
            {
                // if equal then return true;
                return true;
            }
            return false;
        }
        // overrides the virtual method
        // this method is only ran if the word has already been found in the board.
        // therefore, not if statements were required.
        public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,]
hboard)
        {
            // loops for the length of array
            for (int a = 0; a < arr.Length; a++)
            {
                // sets the boolean values on bool board to be true where the word was found
                hboard[startRow, startCol+a] = true;
            }
            return hboard;
        }
    }
}
```

Search Down (Child class):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Chejiaha_Midterm
{
    class SearchDown : Search // searchDown inherits Search
    {

        // overrides virtual method from abstract class
        public override bool searchWord(int startRow, int startCol, char[] arr, char[,] board)
        {
            char[] match = new char[arr.Length];
            // iterate through the array to find match
            for (int a = 0; a < arr.Length; a++)
            {
                // starts at position where first letter was found, then iterates across (static row coordinate)
                // to check if the rest of word matches the characters on the board
                if ((arr[a] == board[startRow + a, startCol]))
                {
                    // assign temp array values
                    match[a] = arr[a];
                }
            }
            // check if the array stored is equal to the word user entered
```

```
            if (match.SequenceEqual(arr))
            {
                // if equal then return true;
                return true;
            }
            return false;
        }

        // overrides the virtual method
        // this method is only ran if the word has already been found in the board.
        // therefore, not if statements were required.
        public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,] hboard)
        {
            // loops for the length of array
            for (int a = 0; a < arr.Length; a++)
            {
                // sets the boolean values on bool board to be true where the word was found
                hboard[startRow + a, startCol] = true;
            }
            return hboard;
        }
    }
}
```

Search Diagonal Down (Child class):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Chejiaha_Midterm
{
    class SearchDiagonalDown: Search // searchDown inherits Search
    {
        // overrides virtual method from abstract class
        public override bool searchWord(int startRow, int startCol, char[] arr, char[,] board)
        {
            char[] match = new char[arr.Length];
            // iterate through the array to find match
            for (int a = 0; a < arr.Length; a++)
            {
                // starts at position where first letter was found, then iterates diagonally +a to
both row and col
                // to check if the rest of word matches the characters on the board
                if ((arr[a] == board[startRow + a, startCol + a]))
                {
                    // assign temp array values
                    match[a] = arr[a];
                }
            }
            // check if the array stored is equal to the word user entered
            if (match.SequenceEqual(arr))
```

```
            {
                // if equal then return true;
                return true;
            }
            return false;
        }

        // overrides the virtual method
        // this method is only ran if the word has already been found in the board.
        // therefore, not if statements were required.
        public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,]
hboard)
        {
            // loops for the length of array
            for (int a = 0; a < arr.Length; a++)
            {
                // sets the boolean values on bool board to be true where the word was found
                hboard[startRow + a, startCol + a] = true;
            }
            return hboard;
        }
    }
}
```

Search Diagonal Up (child class):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Chejiaha_Midterm
{
    class SearchDiagonalUp : Search // searchDown inherits Search
    {
        // overrides virtual method from abstract class
        public override bool searchWord(int startRow, int startCol, char[] arr, char[,] board)
        {
            char[] match = new char[arr.Length];
            // iterate through the array to find match
            for (int a = 0; a < arr.Length; a++)
            {
                if (startRow - a >= 0)
                {
                    // starts at position where first letter was found, then iterates diagonally -a
to row and +a to col
                    // to check if the rest of word matches the characters on the board
                    if ((arr[a] == board[startRow - a, startCol + a]))
                    {
                        // assign temp array values
                        match[a] = arr[a];
                    }
                }
```

```
            }
            // check if the array stored is equal to the word user entered
            if (match.SequenceEqual(arr))
            {
                // if equal then return true;
                return true;
            }
            return false;
        }
        // overrides the virtual method
        // this method is only ran if the word has already been found in the board.
        // therefore, not if statements were required.
        public override bool[,] highlightedBoard(int startRow, int startCol, char[] arr, bool[,]
hboard)
        {
            // loops for the length of array
            for (int a = 0; a < arr.Length; a++)
            {
                // sets the boolean values on bool board to be true where the word was found
                hboard[startRow - a, startCol + a] = true;
            }
            return hboard;
        }
    }
}
```

Layout xaml:

Change font size, margin and padding of the labels in grid.

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

   <Style x:Key="btnStyle">
      <Setter Property="Button.Margin" Value="5,10" />
      <Setter Property="Button.Foreground" Value="White" />
      <Setter Property="Button.FontSize" Value="8"/>
      <Setter Property="Button.Padding" Value="5" />
   </Style>
</ResourceDictionary>

<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                xmlns:local="clr-namespace:Chejiaha_Midterm">
   <Style x:Key="btnStyle">
      <Setter Property="Button.Margin" Value="5,10" />
      <Setter Property="Button.Foreground" Value="White" />
      <Setter Property="Button.FontSize" Value="16"/>
      <Setter Property="Button.Padding" Value="5" />
   </Style>
</ResourceDictionary>
```

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                xmlns:local="clr-namespace:Chejiaha_Midterm">
    <Style x:Key="btnStyle">
        <Setter Property="Button.FontSize" Value="12"/>
    </Style>
</ResourceDictionary>
```

Main xaml:

```
<Window x:Class="Chejiaha_Midterm.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:Chejiaha_Midterm"
        mc:Ignorable="d"
        Title="MainWindow" Height="650" Width="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="731*"/>
            <ColumnDefinition Width="0*"/>
            <ColumnDefinition Width="69*"/>
        </Grid.ColumnDefinitions>
        <DockPanel LastChildFill="False" Grid.ColumnSpan="3">
            <Label Content="Word Search" DockPanel.Dock="Top"
HorizontalAlignment="Center" FontSize="20" />
            <Label Content="Created by Jiahao Chen. PROG32356"
DockPanel.Dock="Bottom" Background="LightSteelBlue"/>


            <StackPanel DockPanel.Dock="Left" Background="Cornsilk">
                <Label>Enter length</Label>
                <TextBox x:Name="length"></TextBox>
                <Button Margin="10" Click="displayBoardBtn">Draw Board</Button>
                <Label>Enter word</Label>
                <TextBox x:Name="str"></TextBox>
                <Button Margin="10" Click="SearchBtn">Search Word</Button>
```

```
            <Label Style ="{DynamicResource btnStyle}" Content="Words found:"/>
            <Label x:Name="lblAcross" Content="Across: 0"/>
            <Label x:Name="lblDown" Content="Down: 0"/>
            <Label x:Name="lblDiagonalUp" Content="Diagonal Up: 0"/>
            <Label x:Name="lblDiagonalDown" Content="Diagonal Down: 0"/>
            <CheckBox x:Name="chkHighlight" Content="Highlight" Checked="checkBox"
Unchecked="unChkHighlight"/>
            <RadioButton Checked="rbLayout1_Clicked">Smaller</RadioButton>
            <RadioButton Checked="rbLayout12_Clicked">Bigger</RadioButton>
            <RadioButton Checked="rbLayout3_Clicked">Normal</RadioButton>
        </StackPanel>
        <StackPanel DockPanel.Dock="Left" Background="Azure" Width="699">
            <Grid Style ="{DynamicResource btnStyle}" x:Name="wordGrid"></Grid>
        </StackPanel>
    </DockPanel>
  </Grid>
</Window>
```