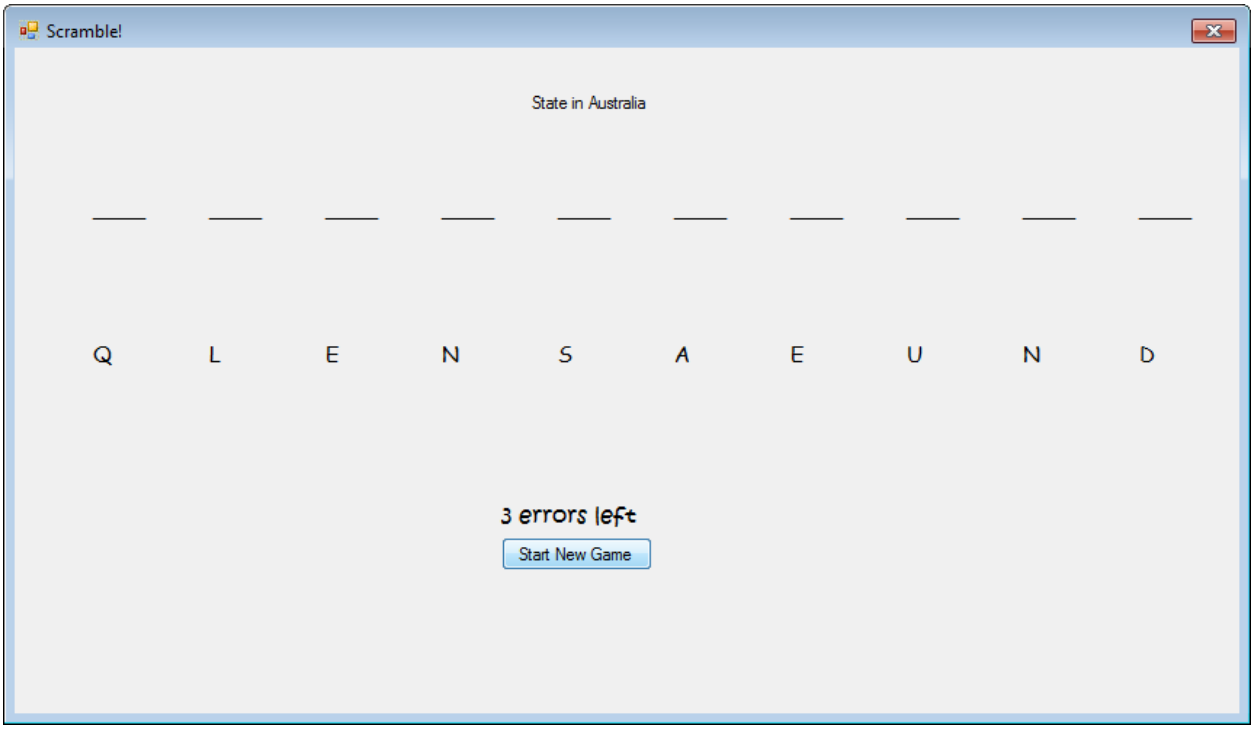
CIS 345 – Business Information Systems Development II – Fall 2014

In-class Exercise 12: Word Game

Due on Blackboard: Today, Monday, November 17 by 10 PM



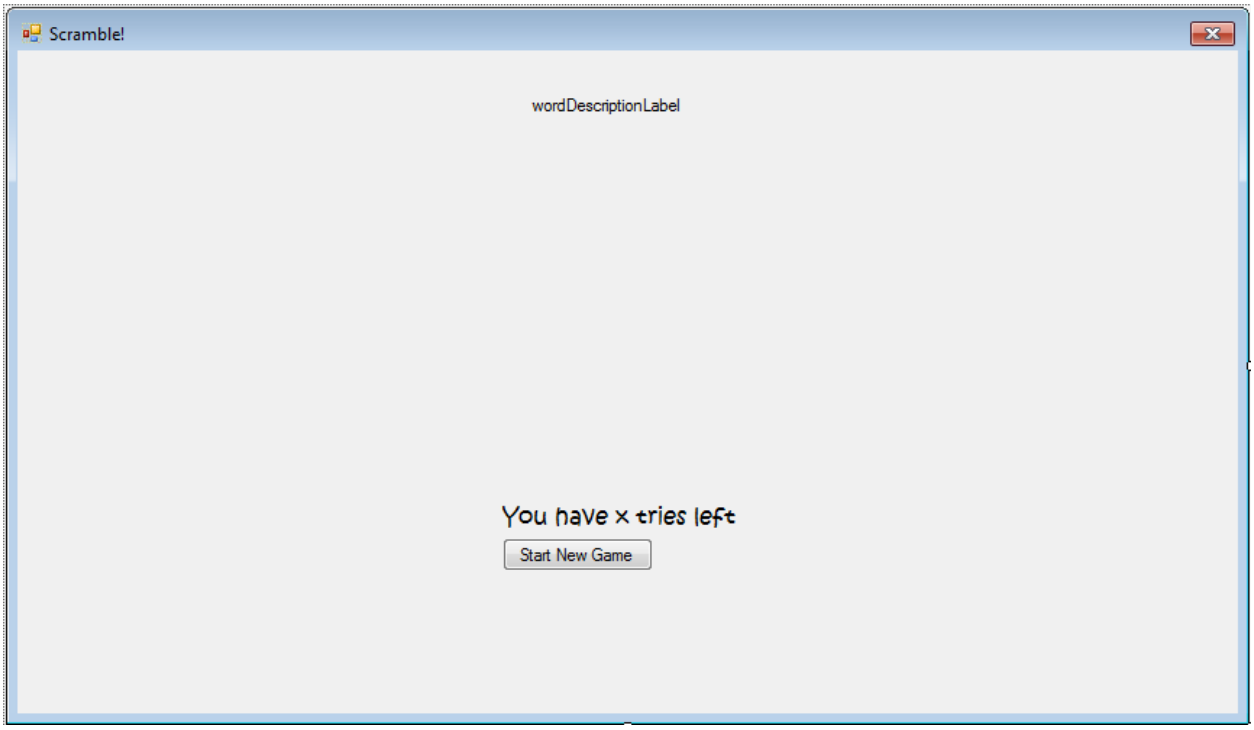
Create a form application that creates Labels dynamically on the fly along with adding event handlers as well as implementing drag/drop mouse behavior. In the process, implement a simple word game that mimics the popular Jumble game. Utilize a DLL file to generate the random word.

Project Details and Requirements

A sample executable file is uploaded and available on Blackboard for you to test program behavior.

Project Name: Scramble
Form Name: Scramble

Form Scramble [Design]



Properties: Since Scramble inherits from Form, a host of inherited properties are already available. The following properties will need to be set at design time using the IDE. Modify your form to look like that in the sample above.

- Text (this property sets the form title): "Scramble!"
- FormBorderStyle: FixedSingle
- MaximizeBox: False
- MinimizeBox: False
- StartPosition: CenterScreen
- Size: 850 * 500

1) Set up the Form as show in the figure above. Use suffixes within the control names so that when you work with in code, you can easily distinguish a TextBox from a Label. e.g. “guessTextBox” should easily be interpreted as the TextBox which contains a guess, while “guessLabel” should easily suggest that is the label accompanying a guess.

Using the IDE, create the following controls as seen in the picture above.

- One Label, called wordDescriptionLabel
- One Label, called errorsLeftLabel
- One Button, called newGameButton

Form Scramble [Scramble.cs Partial Class]

Do not add any code in the Designer.cs or Program.cs classes.

1) DLL Reference

Copy the given DLL and XML files to your project bin\debug folder and load a reference to it in the project. Look up the namespace used in the DLL project using Object Browser. Add that namespace to the Scramble.cs class at the top with all the other "using" statements. The namespaces are indicated in Object Browser using the curly braces, i.e. "{ }"

2) Instance variables

Within the Scramble.cs partial class, add private ***instance*** variables for the following.

- An array of strings called scrambledWordArray
- An array of strings called unScrambledWordArray
- An array of Labels called sourceLabels ←*Labels being dragged*
- An array of Labels called destinationLabels ←*Where the labels are dropped*
- A JumbleCreator object called Jumbler ← *JumbleCreator class is in the DLL*
- An integer, numberOfErrorsLeft, initialized to 3.

There are two sets of variables above: a string array and a label array each for scrambled and unscrambled representations of a word. The JumbleCreator object will expose functionality to generate word jumbles and it will also have methods to get both scrambled and unscrambled versions of words.

METHODS:

Methods should go inside the class as with all objects. The precise location (top, bottom, etc.) does not matter so long as methods are located within the class block.

3) Create a Load event handler for the Form. Within the Load event handler, set both wordDescriptionLabel and errorsLeftLabel to be not visible. *This changes the state of the control even before the user sees the form.*

NOTE: Event handlers for <u>labels</u> are going to be created <u>manually</u>, not using the Designer.

4) Create a method called SourceLabels_MouseDown

Purpose: This method will serve as the event handler for the Source labels. Accordingly, it should have the same parameters as any event handler that is autogenerated.

Parameters: "sender", of type object, and "e" of type MouseEventArgs
Return type: void

General Method logic:

- Declare a temporary label called tmpLabel. Assign the sender object available as a parameter variable to tmpLabel. Since "sender" is actually a Label, cast it as a Label by using parentheses. *You have cast integer variables into doubles before. This is done in the same way except that the target data type now is Label.*
- Call the DoDragDrop method of tmpLabel. Provide it with two arguments. The first argument should be tmpLabel (which is a reference to the actual Label instance). *Here instead of sending the text as part of the drag operation, we are sending the actual Label. Doing this will allow the recipient to access and change all properties of the Label.* The second argument should be DragDropEffects.Move. *This initiates the drapdrop behavior.*

5) Create a method called DestinationLabels_DragEnter

Purpose: This method will serve as the event handler for the Destination labels. According it should have the same parameters as any event handler that is autogenerated.

Parameters: "sender", of type object, and "e" of type DragEventArgs
Return type: void

General Method logic:

- Declare a boolean isLabel
- Assign to isLabel the return value of the GetDataPresent method. The GetDataPresent method is available within the Data object of the "e" object available as a parameter variable. So, first access "e", then Data, and then call the method. Give "typeof(Label)" as the argument to the method. *This means that we are sending it the name of the data type. Accordingly GetDataPresent will check to see if there is anything of data type Label available, and if there is, it will return true.*

- If isLabel is true
    - Set the effect property of "e" to DragDropEffects.Move.

6) Create a method called DestinationLabels_DragDrop

Purpose: This method will serve as the event handler for the Destination labels. According it should have the same parameters as any event handler that is autogenerated. *You are writing this method yourself – not using the IDE to generate it!*

Parameters: "sender", of type object, and "e" of type DragEventArgs
Return type: void

General Method logic:

- Declare a temporary label called tmpDestination. Assign the sender object available as a parameter variable to tmpDestination. Since "sender" is actually a Label, cast it as a Label by using parantheses. *You have cast integer variables into doubles before. This is done in the same way except that the target data type now is Label.*
- Declare a temporary label called tmpSource. Assign to tmpSource the return value of the GetData method. The GetData method is a method available within the Data object inside the "e" parameter variable. So, access, "e", then "Data" and then call the method. The method will require an argument – use "typeof(Label)" as the argument. Finally make sure you cast the return value as a Label. *This takes out the data of type Label, which has been encapsulated inside the Data object.*
- Compare the Tag property of tmpDestination with the Text property of tmpSource. You will need to cast the return value of the Tag property as a string.

If both those two strings are equal

  - Set the Text property of tmpDestination to the Text property of tmpSource. *Essentially, the answer to a blank label is contained inside its tag property. If that answer is the same as the data sent over inside the dragdrop object, use the answer. If it is not, then the user tried to put the wrong letter into place and you don't need to do anything!*
  - Set the visible property of tmpSource to false. *If the user dragged the correct source label onto the correct destination label, then we are setting the source to invisible, thereby taking off that label as a possible future choice.*

Else

  - Decrement the number of errors left by 1.
  - Set the Text of the errors left Label to show the updated number of errors left. You will need to use string concatenation OR the Format method of the String class if you want to use placeholders.
  - If the number of errors left is zero
    - Update the errors left label to tell the user that he/she lost the game.
    - Loop through the destinationLabels array from 0 to its length
      - For each label, set the enabled property to false.

7) LoadWords

Create a method called LoadWords. It should not return anything.

Purpose: The purpose of the LoadWords method will be to create an instance of JumbleCreator, call the methods from JumbleCreator to create a random word and present the scrambled word on the screen as well as blank tiles. It will create labels dynamically as well as assign event handlers for them.

General Method Logic:

- Create a new instance of JumbleCreator and assign the reference to your JumbleCreator variable.
- Call the GetScrambledWord method of the JumbleCreator object. Store the array it returns in your ScrambledWordArray variable. *This is the word the user is going to guess, scrambled.*
- Call the GetUnScrambledWord method of the JumbleCreator object. Store the array it returns in your unScrambledWordArray variable. *This is the unscrambled answer.*

- Set the number of errors left to 3.
- Set the errorsLeftLabel to show the value of the numberOfErrorsLeft variable. Use string concatenation to create a string to be displayed.
- Set the ForeColor of the errorsLeftLabel to Black.
- Make the errorsLeftLabel visible.

- Call the GetWordDescription method of the JumbleCreator object. Store its return value so that it is displayed in the wordDescriptionLabel label. *This is the "question", e.g. "Movie"*
- Set the Autosize property of wordDescriptionLabel to true
- Set the wordDescriptionLabel to visible.

- Declare an integer arrayLength. Put in it the length of the ScrambledWordArray. *ArrayLength now stores how many characters are in the word that the user will guess.*
- Allocate elements to the two arrays of labels you declared earlier. Both the array labels should have *arrayLength* number of elements. *This means creating the array of labels – previously the array variables merely contained null since we didn't know how many labels we would actually need.*
- Declare an integer labelLeftEdge and initialize it to 50.

LoadWords Method continues on the next page →

- Loop through from 0 up to arrayLength. *This loop will create the "source" labels at the bottom of the screen.*
    - Declare and instantiate a new Label called tmpLabel.
    - Set the Font property of tmpLabel: create a new instance of the class Font and provide its constructor with two arguments. The first argument will be the name of a font as a string, e.g. "Come Sans MS". The second argument will be the size of the font, e.g. 11. Assign the resulting reference to the Font property of tmpLabel.
    - Put inside the label's Text the contents of the ScrambledWordArray at the current index location. *Each successive label will therefore show one letter from the string array.*
    - Set the label's Autosize property to true
    - Set the lable's Left property to labelLeftEdge
    - Increment labelLeftEdge by 80
    - Set the label's Top property to 200
    - Set the label to be visible.

    - Tell the label that you want a new event handler for the label's MouseDown event. It should be in the structure shown below (you need to modify it). Use your SourceLabels_MouseDown method as the name of the event handler method.

        ```
        LabelName.MouseDown +=  new
        MouseEventHandler(this.EventHandlerMethodName);
        ```

    - Add tmpLabel to the Controls collection of the form using the Add method.
    - Assign tmpLabel to the sourceLabels array at the current index location.

    These two lines add the tmpLabel to the Controls collection as well as our array of Labels. In the future, we could get to the Label using either of the two references.

- Reset labelLeftEdge to 50.

LoadWords Method continues on the next page →

- Loop through from 0 up to arrayLength. *This loop will create the "destination" labels at the top of the screen.*
    - Declare and instantiate a new Label called tmpLabel.
    - Set the Font property of tmpLabel: create a new instance of the class Font and provide its constructor with two arguments. The first argument will be the name of a font as a string, e.g. "Come Sans MS". The second argument will be the size of the font, e.g. 11. Assign the resulting reference to the Font property of tmpLabel.
    - Put a series of underscores inside the label's Text property
    - Set the label's Autosize property to true
    - Put the string found in the current index location of the UnScrambledWorArray inside the Tag property of the label. *The Tag property is like an empty variable that can be utilized to store strings – all Controls tend to have a Tag property. This will allow the "answer" to be hidden within every label while the Text property shows merely a big underscore to the user.*

    - Set the Top property of the label to 100.
    - Set the Left property of the Label to labelLeftEdge
    - Increment labelLeftEdge by 80

    - Set the label's AllowDrop property to true

    - Tell the Label that you want a new event handler for the DragDrop method. It will be in the structure shown below. Use the method DestinationLabels_DragDrop as the name of the event handler method.

```
LabelName.DragDrop += new DragEventHandler(this.EventHandlerMethodName);
```

    - Tell the Label that you want a new event handler for the DragEnter method. It will be in the structure shown below. Use the method DestinationLabels_DragEnter as the name of the event handler method.

```
LabelName.DragEnter += new DragEventHandler(this.EventHandlerMethodName);
```

    - Add tmpLabel to the Controls collection of the form using the Add method
    - Assign tmpLabel to the current index location of the destinationLabels

      Assigning tmpLabel to these two different location creates TWO references to the same Label. We could later access the label either from the label array OR from the Controls collection of the form. In fact, tmpLabel is a THIRD reference to the same label. However, this third reference is only temporary and will be lost after this loop ends.

8) Click event handler for the newGameButton

Purpose: The Click event handler method is called when the clicks the button. This method will reset all the existing references to arrays and Controls, etc. and call LoadGame which starts the game. It is important to reset the references first to ensure that when the user presses while a game is already running, the existing game is "stopped" and then a new one is loaded.

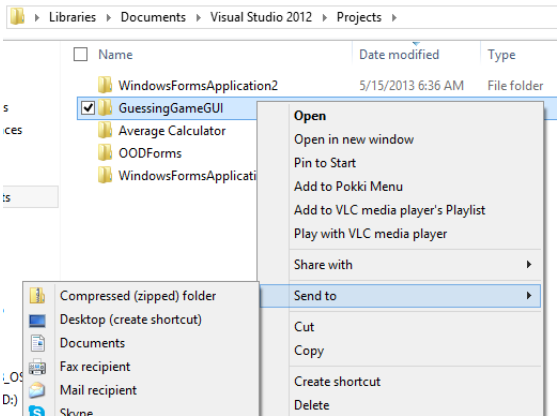
General Method logic:

- Loop through all the Controls using a for loop. However, do not loop from 0 to Controls.Count. Rather start from Controls.Count – 1 and count down while your counter is greater than or equal to zero. *This fixes a problem that happens when the Control count changes whenever a control is deleted.*
    - If the control at the current index is a Label (use the *is* keyword and the name of the data type, e.g. "xyz is Label"*) AND* the Name property of the control at the current index is NOT "wordDescriptionLabel" AND the Name property of the control is not "errorsLeftLabel"
        - Dispose of the current Control by call its Dispose() method. *You will still need to use the index number of the Controls collection to access the control at the current index.*
- Set all your five instance variables (four arrays and one JumbleCreator instance) to null
- Set the errorsLeftLabel to invisible.
- Call LoadWords.

Submission Instructions

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to ***zip the entire project folder*** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \ProjectName\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
    - e.g. " // In-class 12, Jane C. Smith, CIS 345, Tuesday 9:00 AM"
    - Your project is called Paint
- Zip filename is: FirstNameLastName_Inclass12.zip

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.
- Make sure you have submitted your file and not just saved a draft on Blackboard. A blue clock indicates a submission in progress, i.e. a draft, not a submission.