Class Assignment – Card Games

Specification Part B – **GUI Layouts**

Part B weighting: 7.5% of 40%
Parts A and B Due: Thursday, September 13th during your usual tutorial

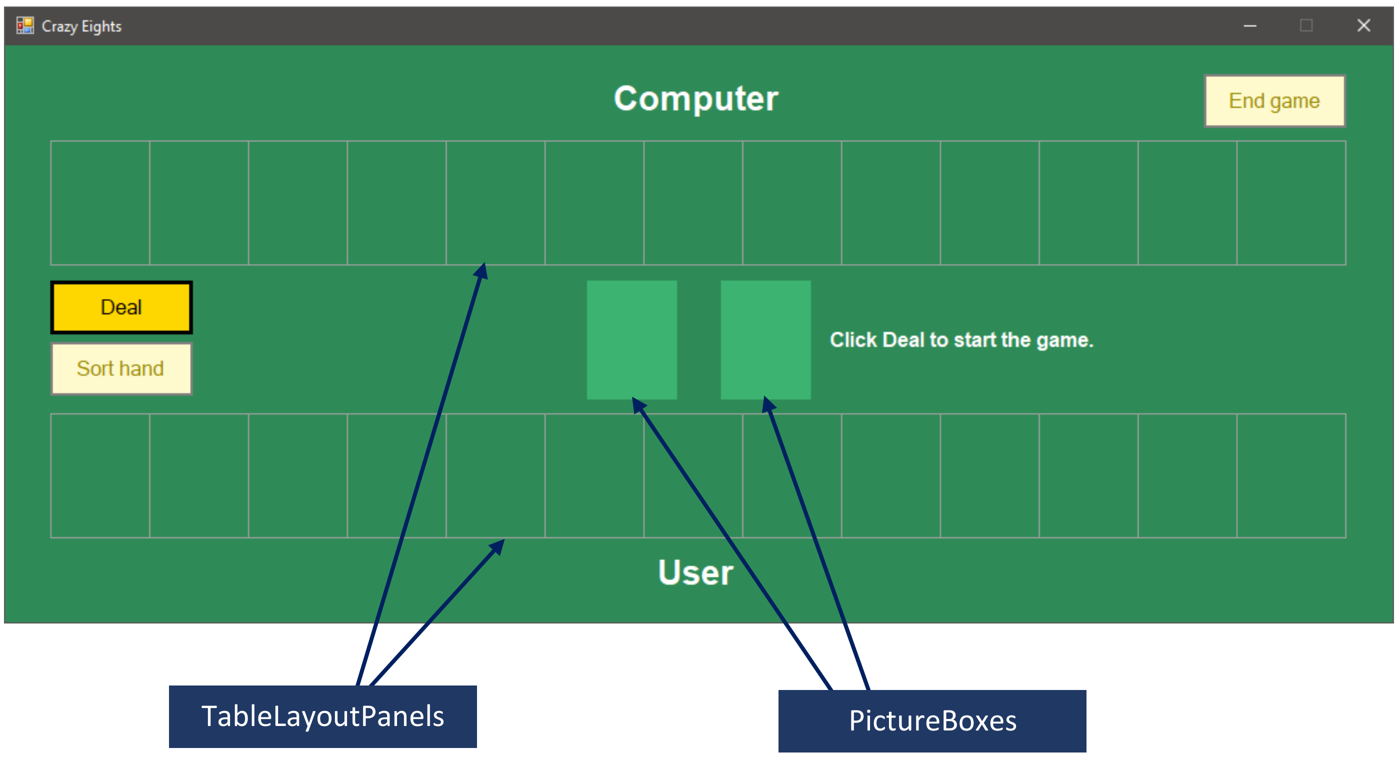
Part B Overview

In this part of the assignment, you will implement the layout of all required **Windows Forms** and some important GUI functionality.

Your code for this part will be assessed with the final submission. However, in the tutorial, you must demonstrate that all the required form components are in your GUI layouts for **Crazy_Eights_Form** and **Choose_Suit_Form**. You must also demonstrate that Card Images can be displayed, and a Suit can be selected via the **Choose_Suit_Form** and the result can be received by the **Crazy_Eights_Form**. These important functionalities are crucial to beginning on part C.

Crazy_Eights_Form

The **Crazy_Eights_Form** should have a layout like the screenshot below. It does not need to be perfect or have exactly the same visual characteristics as the screenshot. It is only important that you have used the correct components with appropriate names, and that they are easily readable and sensibly positioned.



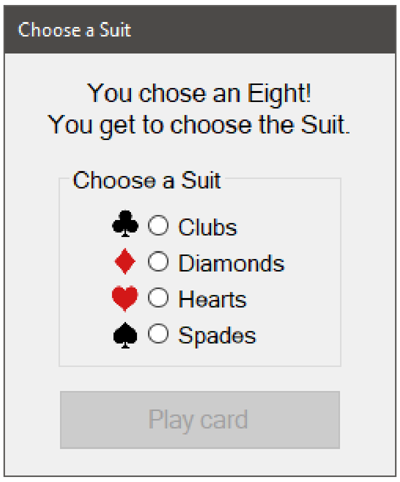
Suggested components and properties

The **Crazy_Eights_Form** must have each of the components below. Feel free to change the visual characteristics of the form components. The suggestions below are a good starting point.

Form	Text	“Crazy Eights”
	Name	Crazy_Eights_Form
Computer label	Text	“Computer”
	Font	[you decide]
	Name	[choose an appropriate name]
User label	Text	“User”
	Font	[you decide]
	Name	[choose an appropriate name]
TableLayoutPanel (Computer) **TableLayoutPanel (User)**	ColumnCount	13
	RowCount	1
	Columns	Set each column to occupy 7.69% of the width
	Size	1036, 100 [suggestion]
	Name	[choose an appropriate name]
	AutoSize	False
	Anchor	Left, Right [if you want it to resize]
PictureBox (Left - DrawPile) **PictureBox (Right - DiscardPile)**	Size	72, 95 [suggestion]
	Name	[choose an appropriate name]
Instruction label	Text	“Click Deal to start the game.”
	Name	[choose an appropriate name]
Deal button	Text	“Deal”
	Name	[choose an appropriate name]
Sort hand button	Text	“Sort hand”
	Name	[choose an appropriate name]
	Enabled	False
End game button	Text	“End game”
	Name	[choose an appropriate name]

Choose_Suit_Form

The **Choose_Suit_Form** is a small form used to ask the user to choose a Suit. In Crazy Eights, when an Eight of any suit is played, the player can change the suit of their card. The **Choose_Suit_Form** should have a layout like the screenshot to the right. Create the **Choose_Suit_Form** in the **GUI** project.



The PictureBoxes are optional. If you wish to add PictureBoxes, the images necessary are in the GUI/Images folder.

Suggested components and properties

You must have each of the components below. Feel free to change the visual characteristics of the form components. The suggestions below are a good starting point.

Form	Text	"Choose a Suit"
	Name	Choose_Suit_Form
	FormBorderStyle	FixedDialog
	ControlBox	False
	ShowInTaskbar	False
	TopMost	True
	StartPosition	CenterParent
Instruction label	Text	"You chose an Eight! You get to choose the Suit"
	Font	[you decide]
	Name	[choose an appropriate name]
Choose Suit Groupbox	Text	"Choose a Suit"
	Font	[you decide]
	Name	[choose an appropriate name]
Play Card button	Text	"Play card"
	Name	[choose an appropriate name]
	Font	[you decide]

Displaying Cards in the GUI

This section explains how to display cards in the **Crazy_Eights_Form**.

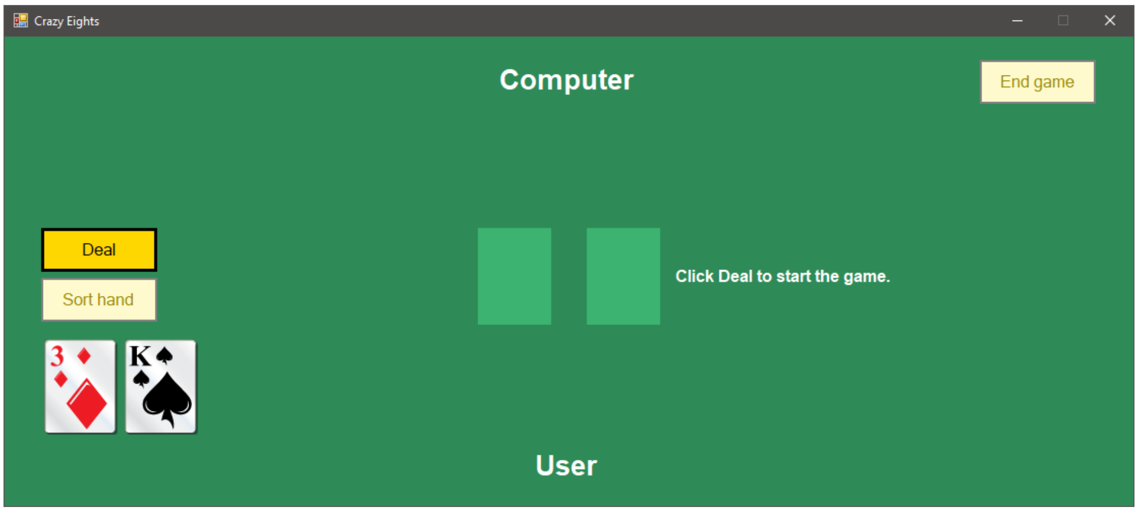
The `DisplayHand` method can display a `Hand` in a `TableLayoutPanel`. If the panel is for the user's hand, it will also add a *click* event handler that calls the `picPlayCard_Click` method when the card picture is clicked. You will need to add in both methods below. The code assumes that the user's `TableLayoutPanel` is named `tblUserHand`. If you called the user's panel something different, you will need to change this.

```
private void DisplayHand(Hand hand, TableLayoutPanel panel) {
    // remove any previous card images
    panel.Controls.Clear();

    // repeat for each card in the hand
    for (int i = 0; i < hand.GetCount(); i++) {
        // add a picture box to the panel with the appropriate image
        PictureBox picCard = new PictureBox();
        picCard.SizeMode = PictureBoxSizeMode.AutoSize;
        picCard.Image = Images.GetCardImage(hand.GetCard(i));
        panel.Controls.Add(picCard, i, 0);

        // add an event handler if it is being added to the user's panel
        if (panel == tblUserHand) {
            picCard.Click += new System.EventHandler(this.picPlayCard_Click);
        }
    }
}

private void picPlayCard_Click(object sender, EventArgs e) {
    // get the picturebox that was clicked
    PictureBox picCard = (PictureBox)sender;

    // determine the position of the picturebox that was clicked
    int columnNum = ((TableLayoutPanel)((Control)sender).Parent).GetPositionFromControl(picCard).Column;

    // ...you will need to continue this yourself in part C...
    MessageBox.Show(string.Format("Clicked column {0}", columnNum)); // temporary
}
```

Here is an example demonstrating how to use `DisplayHand`. This can be written in the form's constructor as a temporary test. The example below assumes the player's `TableLayoutPanel` is called `tblUserHand`.

```
Hand userHand = new Hand(new List<Card> {
    new Card(Suit.Diamonds, FaceValue.Three),
    new Card(Suit.Spades, FaceValue.King)
});
DisplayHand(userHand, tblUserHand);
```

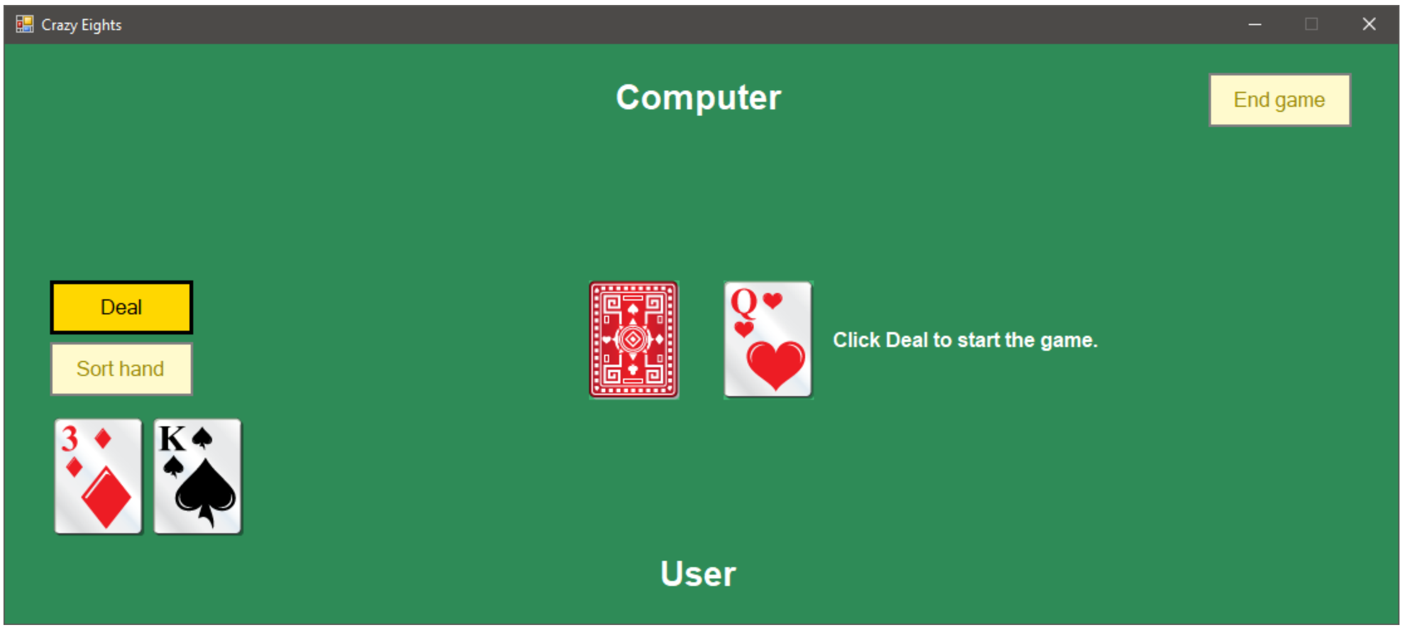
This is the result when the GUI executes the above code:



The following code will add the *card back* image to the draw pile PictureBox, and a *card face* image to the discard pile PictureBox. It assumes the draw pile and discard pile PictureBoxes are called *picDrawPile* and *picDiscardPile*.

```
picDrawPile.Image = Images.GetBackOfCardImage();
picDiscardPile.Image = Images.GetCardImage(new Card(Suit.Hearts, FaceValue.Queen));
```

Executing the code in the constructor should allow you to see a result such as the one below:



Displaying Instructions to the User with a wait time

As the user plays games of Crazy Eights, various instructions will be displayed to explain what is happening in the game. It is important that the user has time to read and understand what is happening between steps, as multiple steps may occur in succession. For this reason, you should implement the method below (change lblInstructions to what you called your instruction label).

```
private void UpdateInstructions(string message, bool wait = false) {
    lblInstructions.Text = message;
    lblInstructions.Refresh();
    if (wait) {
        System.Threading.Thread.Sleep(1000);
    }
}
```

Updating the GUI during wait times

The *Refresh* method used above is available to any GUI component and will be **essential** for updating a component's appearance during wait times before control returns to the user.

The `Sleep` method causes execution to pause for several milliseconds. In the above case, if *wait* is set to true, then the window will pause for 1000 milliseconds (1 second). During this time, no other processing on the form will occur. This is useful if several events occur in succession – the user will have time to see what is happening.

Getting a Suit from the User

To create and show the **Choose_Suit_Form** via the **Crazy_Eights_Form**, use the following code. You may wish to test this in a *click* event handler for the *Deal* button as a temporary test for now.

```
Choose_Suit_Form suitForm = new Choose_Suit_Form();
suitForm.ShowDialog();
```

The `ShowDialog` method will pause execution of the event handler until the **Choose_Suit_Form** has closed. It will also prevent the user from going back to the **Crazy_Eights_Form** without choosing a Suit first.

To close the **Choose_Suit_Form** after the user selects a suit and clicks the *Play card* button, use the following:

```
this.Close();
```

To find out which Suit the user chose after the form has closed, write code in the **Crazy_Eights_Form** to access a public method from **Choose_Suit_Form**. You may wish to complete a method to **Choose_Suit_Form** such as the one below, to get the user's chosen Suit. An additional private variable in **Choose_Suit_Form** may be necessary to achieve this.

```
public Suit GetChosenSuit() {…}
```

To help demonstrate this functionality, and the functionality of the `UpdateInstructions` method, add the following code after you determine the Suit that was chosen:

```
UpdateInstructions(string.Format("You chose {0}.", chosenSuit.ToString()), wait=true);
UpdateInstructions("Click Deal to start the game.");
```

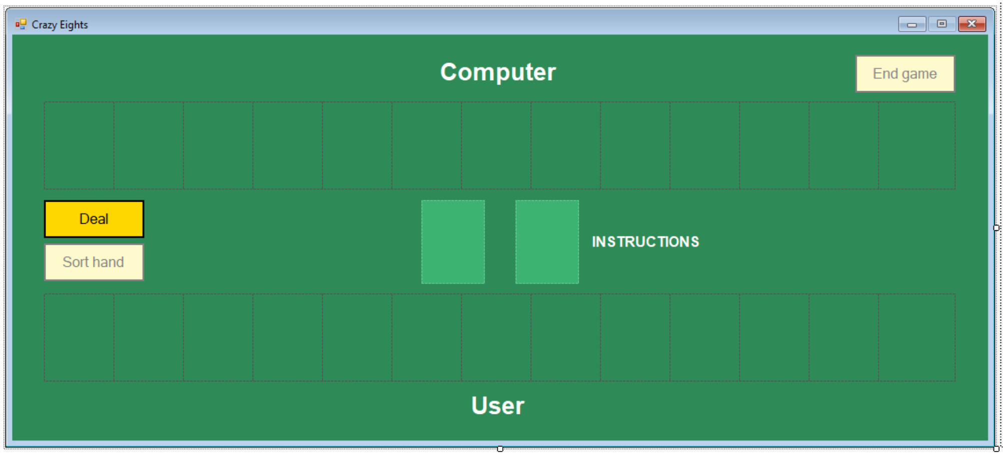
This should update the instruction label so that it first reads '*You chose Hearts*' (for example). A second or so should then pass, and the instruction label should update to read *'Click Deal to start the game.'*

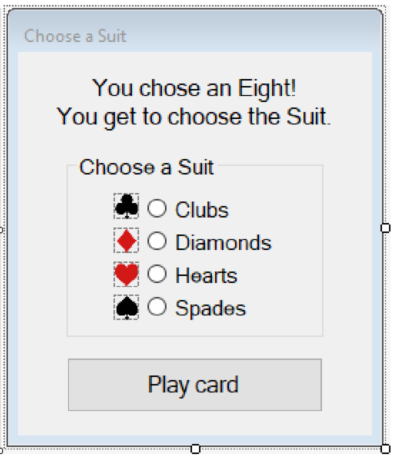
Demonstrations

To gain full marks for part B, you need to perform the following demonstrations. **See the Part B demonstration video on Blackboard if you are unsure.**

Demonstration 1 – Complete GUI Layouts

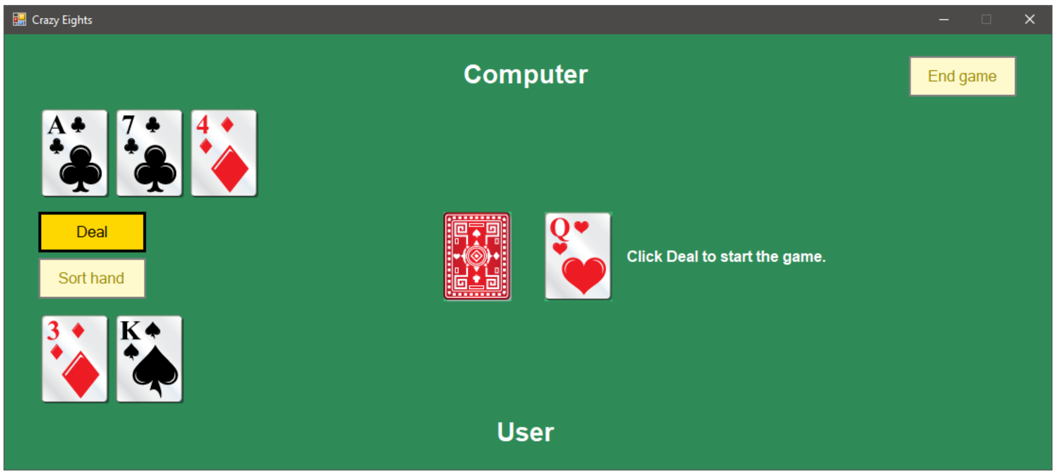
Your GUI layouts should be complete and contain the correct components. They do not need to look perfect – only readable and positioned sensibly.





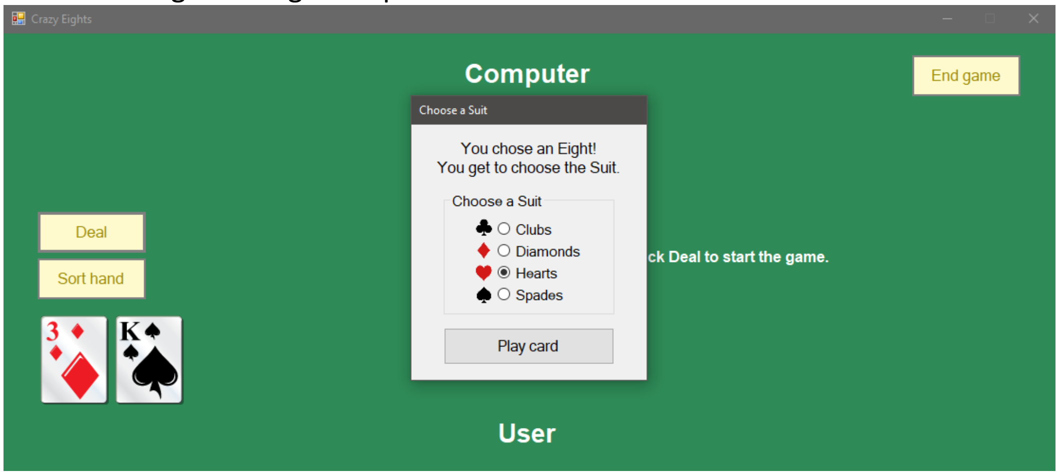
Demonstration 2 – Displaying cards in each of the main areas

You need to demonstrate that you can display Card images in each area (the two PictureBoxes and the two TableLayoutPanels).



Demonstration 3 – Opening the Choose_Suit_Form (via *Deal* or similar)

You need to be able to open the **Choose_Suit_Form** via the **Crazy_Eights_Form** (you can do this via the *Deal* button for now if you wish).



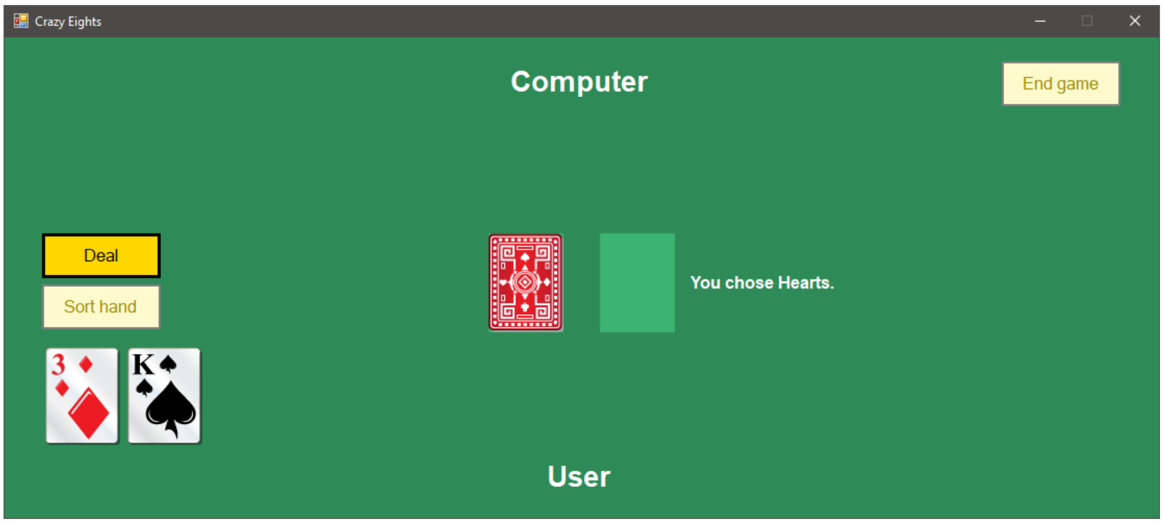
You chose an Eight!
You get to choose the Suit.

Choose a Suit
○ Clubs
○ Diamonds
◉ Hearts
○ Spades

ck Deal to start the game.

Demonstration 4 – Determining which Suit was chosen

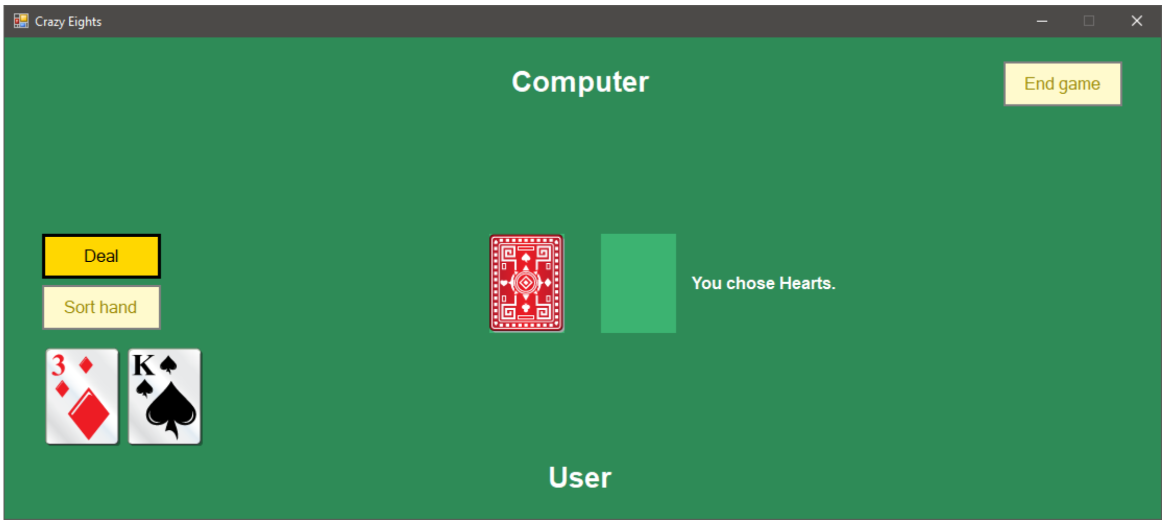
After the user selects a Suit from the **Choose_Suit_Form**, you need to demonstrate that the **Crazy_Eights_Form** can tell which Suit was chosen. You could do this by displaying the chosen suit in the instruction label.



You chose Hearts.

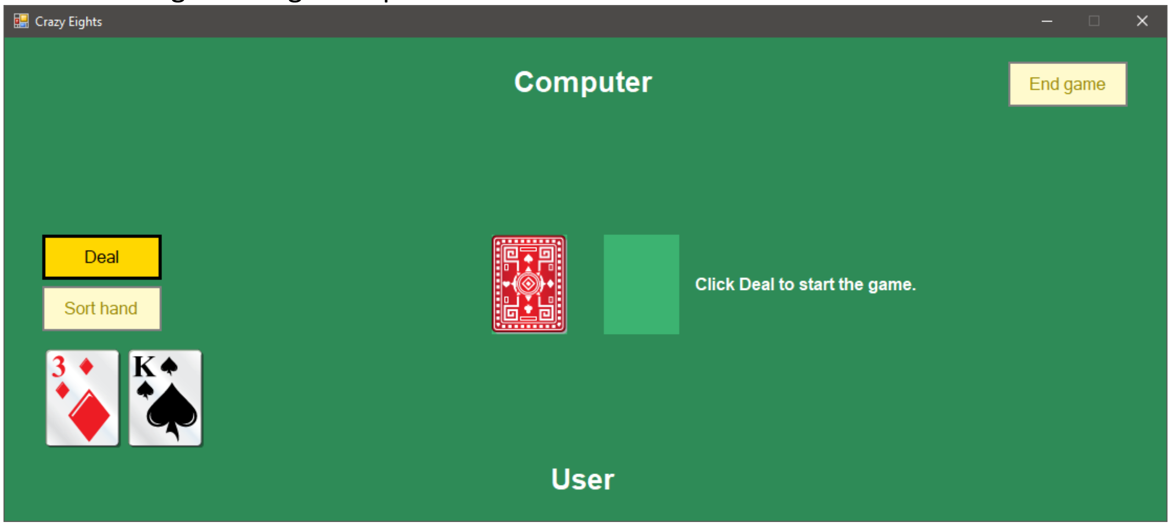
Demonstration 5 – Displaying instructions and waiting

You need to be able to demonstrate that you can update the GUI several times, with a delay between updates. For example, the instruction label is first updated to read “You chose Hearts” after selecting a suit.



You chose Hearts.

Then, a second or so passes, before the instruction label updates again to “Click Deal to start the game.”



Click Deal to start the game.