Class Assignment – Hare and Tortoise
Part C:

Introduction

In this part of the assignment you will continue to add functionality to the Form: the DataGridView will be populated with the player's data, the player's token will be placed onto the Start square and code will be written to move the tokens as a result of throwing of two dice. Finally a simple debugging ability will be implemented.

1 Progressing towards the Start of the Game.

1.1 Populating the DataGridView Control.

In order that the **DataGridView** Control can display the player information at the start of the game, as seen in Part A, page 2 screenshot, additional global variables and some code needs to be added to the **HareAndTortoise_Game** class.

Each player has a default name at the start of the game, **"One", Two", …, "Six"**. Add **a string array** which contains these values as a class variable.

Each player has a colour associated with them which will be the colour of their token on the GUI image of the board. Add **an array of Brush** where the elements are the following colours in the order, **Black, Red, Gold, GreenYellow, Fuchsia and BlueViolet.** The colours are defined in the **Brushes** class and referenced by `Brushes.Black, Brushes.Red, etc.`

Add an integer variable, **numberOfPlayers** and assign the value 6 to it.

Add a method which will initialise each of the players with their corresponding "name" and "token colour" and then add the player to the **BindingList** players. Use the **Start** square of the Board class as the second parameter to the player's constructor.

Call the above method from **SetUpGame()**.

Switch to the **HareAndTortoise_Form (Design),** use the **DataGridView** control to change the **ImageLayout** property of the **Colour** column to **Stretch.** (Comment: This is necessary so that the images completely fill this column's cells.)

In **HareAndTortoise_Form (Code View)** set the **DataSource** property of your **DataGridView** control to the **BindingList i**n **HareAndTortoise_Game.** Place this assignment statement as the last statement within the **HareAndTortoise_Form** constructor.

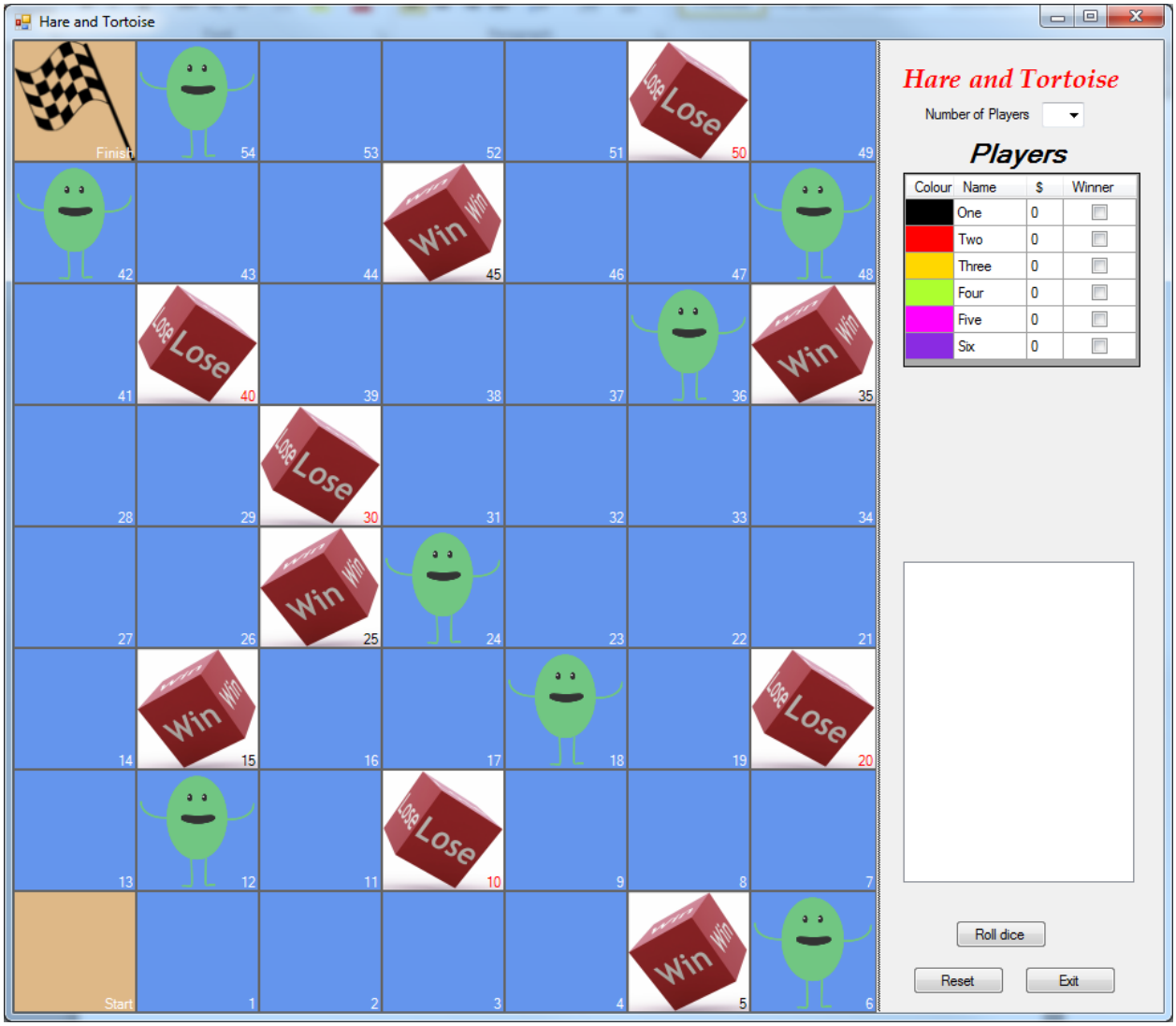
At this point, run your program and you will see the **DataGridView** control containing the six players' data. The control probably has horizontal and/or vertical scroll bars.

To remove the scroll bars make the following adjustments:

First adjust the size of the **ListBox** to approximately 200 by 260. Change the **ListBox Anchor** property to **Bottom, Left, Right.**

Adjust the size of the **DataGridView** to approximaely 195 by 160. Rerun your program. You may have to repeatedly adjust the size of the **DataGridView** control and the **ListBox** in **Design View** to have all of the players' data displayed without scroll bars.

Your GUI board should look similar to the screenshot below.



1.2 Making the Players' Tokens visible on the GUI board.

Each square of the GUI board is a **SquareControl** object and at various times of a game, a square may or may not have one or more players on that square. Look at the **SquareControl** class (Code view), the size and purpose of the boolean array **containsPlayers** should now be apparent.

Develop and implement a method in the **HareAndTortoiseForm** class which will update the square that all of the players are on. The body of this method will implement the following algorithm:

for each player (a for loop not foreach!)
Determine which square the player is on.
Get the SquareControl of that square
Update containsPlayers element which corresponds to this player
end for
Redisplay the GUI Board.

Remember a player object knows what square the player is on and each square is associated with a particular SquareControl object.

You implemented a method which maps a square's number to a position on the **TableLayoutPanel** in **Part A**.

The **TableLayoutPanel** class has **GetControlFromPosition(column, row)** which returns the control occupying the position specified by the row and column.

To ***"Redisplay the GUI Board"*** use the following statement

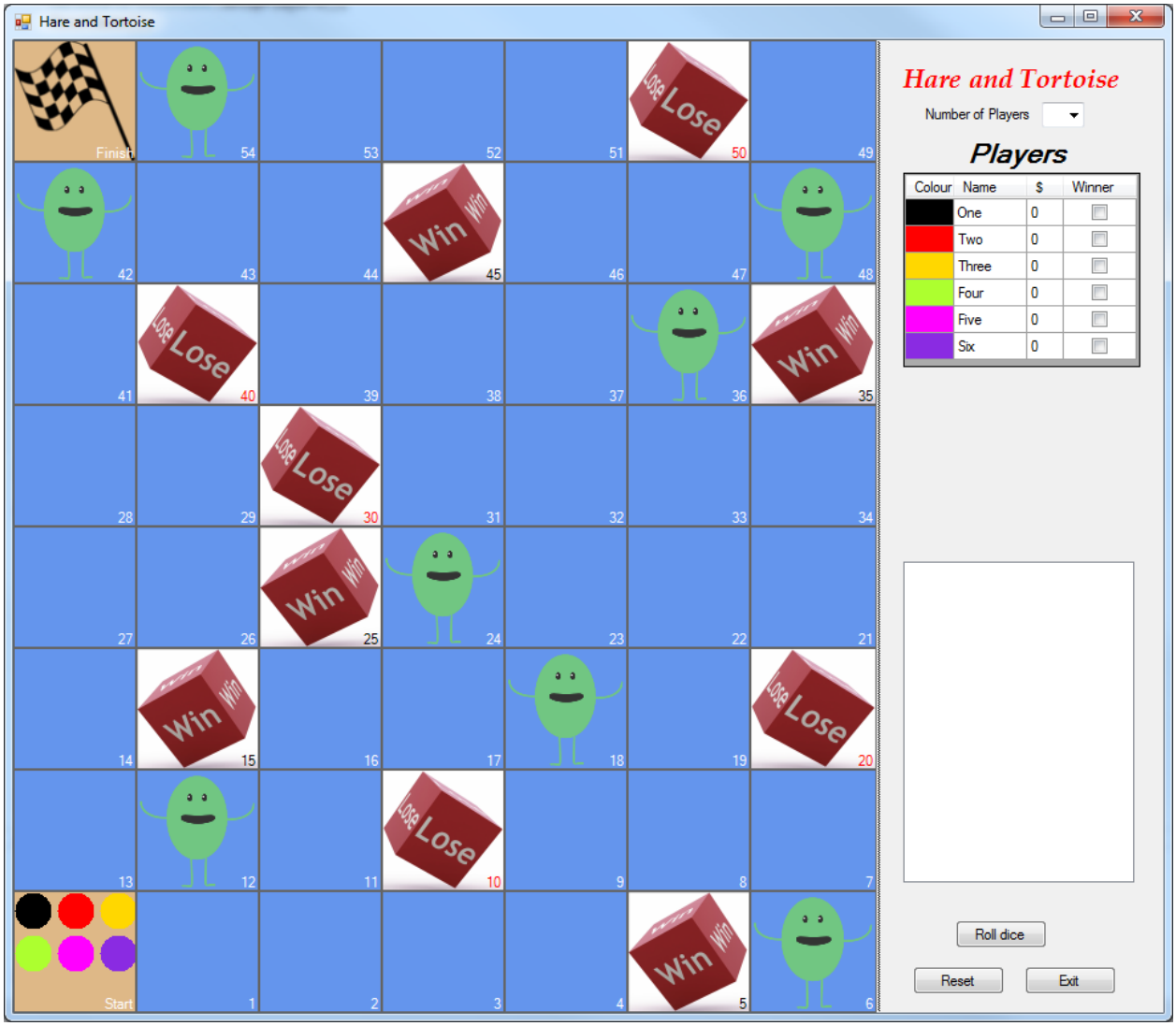
```
gameBoardPanel.Invalidate(true);
```

where **gameBoardPanel** is the "name" of your **TableLayoutPanel**.

The method name, **Invalidate,** seems contradictory. This method causes a "paint" message to be sent to the control (the TableLayoutPanel) and its children (the SquareControls) so the result is that each SquareControl is redrawn.

In the **HareAndTortoiseForm** constructor add a call to this update method that you have just implemented.

Your GUI Board should now have the six tokens on the **Start** square, see screenshot below.



2 General Game Play

Many board games have a common pattern to their play. All players start at the same position. One at a time, each player takes a turn which involves rolling a pair of dice and moving their token the number of squares equivalent to the total shown by the dice. When all players have had a turn, that **round** is complete. If the game is not finished another round commences.

In the Hare and Tortoise game the end of the game will occur when one or more players have reached the **Finish** square. However in this game all players will complete their turn within the round any player reaches the **Finish** square.

The winner will be the player who has the most amount of money at that time, even if that particular player has not yet reached the **Finish** square. In the case of more than one player with the highest amount of money then the winner will be the player who has the highest amount of money and has advanced the furthest on the board. If there is more than one player on that square with the highest amount of money then all of those players are declared winners. Money will be discussed in a future specification.

3 Playing One Round

Implement the event handler for the **"Roll dice"** button. The event handler will call a method in the **HareAndTortoise_Game** class which will **"play one round"**. Upon returning from that method, call the method, to update the squares that the players are on, which you implemented in **1.2** above.

3.1 Play one round in HareAndTortoise_Game

In **HareAndTortoise_Game** playing one round will involve each player **"having their turn"** by calling a method in the **Player** class to roll the two dice.

The two dice will be declared in the **HareAndTortoise_Game** class, and passed to a method in the Player class to **"roll the dice"**.

The **Player** class **"roll the dice"** will roll the dice to determine the number of squares to move the player forward. The player will move to that square by setting the player's **location** to that square. For example, if the player is currently on square number 7 and throws a total of 9, the player will be on square number 16.

3.2 Tokens moving from the Start Square.

If you have successfully implemented the event handler for **Roll dice** button and the subsequent methods in **the HareAndTortoise_Game** and **Player** classes, when you click the **Roll dice** button, you should see the 6 tokens on other squares of the GUI Board.

You will also see the tokens are still on the **Start** square. Clicking **Roll dice** again will move the tokens further along the board but you will still see the tokens on their previous squares as well as the **Start** square.

When moving a player from one square to another, we need to "remove" the player from the **"starting" SquareControl** before we actually move the player and then update the **SquareControl** that they finish on. Two possible implementations exist:

Write a similar method to that in **1.2** which removes the players from their **"starting" Square Control**, only one line would need to be changed.

Alternatively, add a parameter to the method of **1.2** which indicates "adding" or "removing" a player from a **SquareControl** and adjust the logic of the method to either add or remove a player.

The latter is the preferred implementation as it adheres to the **DRY** principle. The former would involve some cutting and pasting which is never a good practice when writing code.

Add a call to this method to remove the players as the first statement of the **Roll dice** event handler. Now run your program. Each token should appear on only one square after clicking the **Roll dice** button. Clicking the button a few times should see all the tokens move along the board towards the **Finish** square.

Appendix – Using Trace to Write Output to a ListBox

The **Trace** class is included in C# to support *tracing* in an application. The **Trace Listener** class determines where the output is sent.

To provide some simple debugging for this assignment, a **Trace Listener class** is provided which sends output to a **ListBox**. This class is complete and you are not expected to understand how and why it works. Instructions to use it along with some coding example are provided below after the instructions to incorporate the **Trace Listener** into your **HareAndTortoise** project folder.

Adding the Trace Listener: Download the file, **ListBoxTraceListener.cs** from Blackboard and copy it to the **HareAndTortoise** project folder of your assignment along with the other cs files, **HareAndTortoiseForm.cs**, **HareAndTortoise_Game.cs,** etc.

In Visual Studio, open Solution Explorer, select the **HareAndTortoise** Project, **Add *"an Exiting Item"***. The File Dialogue window should open on the directory containing **ListBoxTraceListener.cs,** if not **Browse** to locate it.

To both **HareAndTortoiseForm** and **HareAndTortoise_Game** add the directive "**using System.Diagnostics**;"

In the **HareAndTortoiseForm** constructor add as the last statement:

```
Trace.Listeners.Add(new ListBoxTraceListener(listBox));
```

Replace the parameter **listBox** above by the name of the **ListBox** on your Form.

Add the following two methods to the bottom of your **HareAndTortoise_Game** class

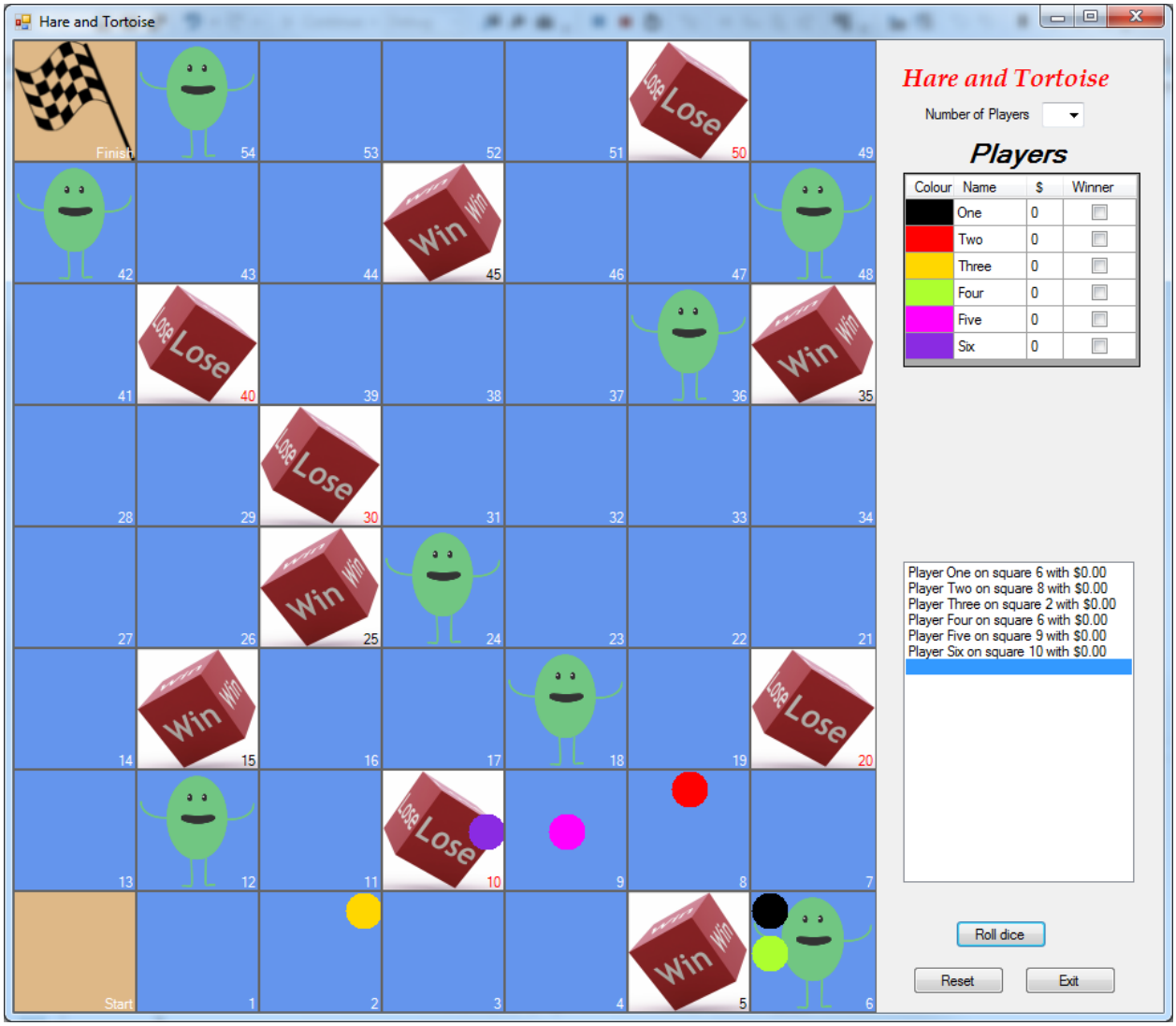
```
public static void OutputAllPlayerDetails() {
   for (int i = 0; i < numberOfPlayers; i++) {
       OutputIndividualDetails(Players[i]);
   }
} // end OutputAllPlayerDetails


/// <summary>
/// Outputs a player's current location and amount of money
/// pre:  player's object to display
/// post: displayed the player's location and amount
/// </summary>
/// <param name="who">the player to display</param>
public static void OutputIndividualDetails(Player who) {
    Square playerLocation = who.Location;
    Trace.WriteLine(String.Format("Player {0} on square {1} with {2:C}",
                                  who.Name, playerLocation.Name, who.Money));
}// end OutputIndividualDetails
```

Add the following method to the **HareAndTortoiseForm** class

```
private void OutputPlayersDetails() {
    HareAndTortoise_Game.OutputAllPlayerDetails();
    listBox.Items.Add("");
    listBox.SelectedIndex = listBox.Items.Count - 1;
}
```

The last two statements ensure that the **ListBox** scrolls down, so you can see the most recent output. To test this additional code, add a call to **OutputPlayersDetails** as the last statement in the **Roll Dice** event handler.

If you have added all of the above code correctly, when you click on the **Roll Dice** button, you should see the **ListBox** with 6 lines of output, similar to this screenshot, reporting the square number that each player's token is on.



Click **Roll dice** a second or third time to see the output in the ListBox.