Class Assignment – Hare and Tortoise Part D:

1 Introduction

At the end of this part of the assignment your program will be play a basic version of the game through to its conclusion. All necessary event handlers will have been implemented.

2 Completing Player.cs

When the game starts each player will have an initial amount of money ($100). Add an extra line to the **Player's constructor** to initialise the **money** instance variable. This is not an additional parameter to the constructor.

Add two public methods to the **Player** class with the following method headings

```
    public void Add(int amount)
```
and
```
    public void Deduct(int amount)
```

Add(…) will increase the player's money by amount.

Deduct(…) will decrease the player's money by amount, however the player's money will never be negative. So if the amount deducted causes the player's money to be a negative value, it is set to zero (0).

3 Adding a method to Square.cs and its subclasses

In the Square class, add the following method

```
    public virtual void EffectOnPlayer(Player who) { }
```

Note this method has and will have an empty body for this assignment. This is no reward or penalty for finishing on an "ordinary" square.

In each of the subclasses of **Square**, will have their own version of this method

```
  public override void EffectOnPlayer(Player who)
```

In the **Win_Square** class the **EffectOnPlayer** method will add $15 to the player's money. (Hint: call the Add method of Player)

In the **Lose_Square** Class the **EffectOnPlayer** method will deduct $25 from the player's money.

In the **Chance_Square** class the **EffectOnPlayer** method will either add or deduct $50 from the player's money. There is 50% chance for either option to occur.

In the **Player Class** where you move the player according to the roll of the dice, add a statement to call the **EffectOnPlayer** method after the player has moved to a new square. For ‘ordinary” squares no change will occur, however for the “special” squares, the player’s money amount will change if you have implemented the desired functionality correctly.

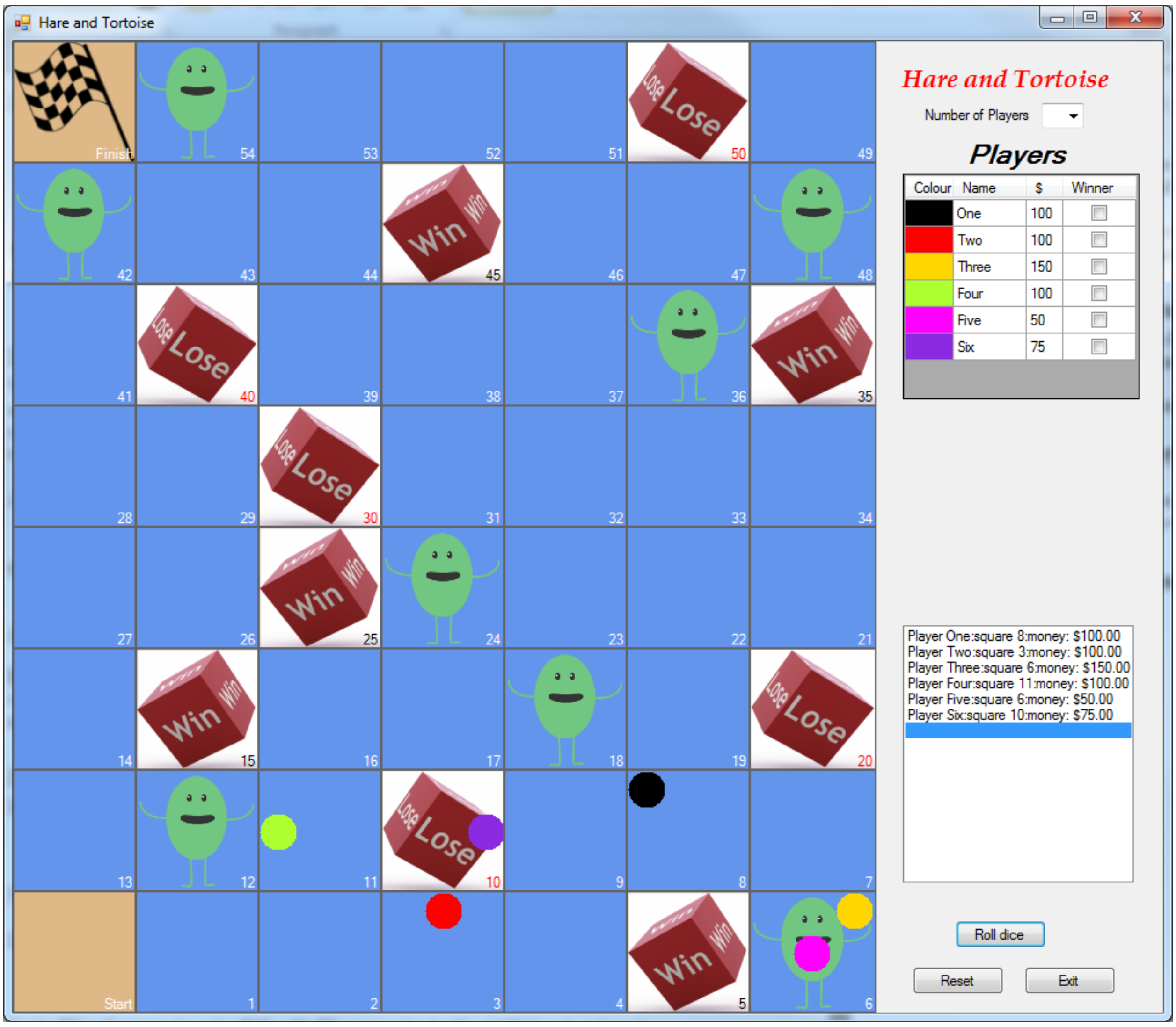
Note at this stage the value of **money** in the **DataGridView** control will not change as the tokens move around the board.

4 Updating DataGridView

Add a void method to **HareAndTortoiseForm** with the name **UpdateDataGridView** which has no parameters. Add the following statement to this method; no other statements are required.

```
        HareAndTortoise_Game.Players.ResetBindings();
```

Call this method at the end of the **Roll Dice** event handler. Run you program, the money column values will be updated when necessary. In the screenshot below, after one round, Player’s Three and Five have landed on a Chance square and won and lost $50 respectively. Player Six finished on a Lose square so has lost $25.



Colour	Name	$	Winner
⬛	One	100	☐
🟥	Two	100	☐
🟨	Three	150	☐
🟩	Four	100	☐
🟪	Five	50	☐
🟪	Six	75	☐

```
Player One:square 8:money: $100.00
Player Two:square 3:money: $100.00
Player Three:square 6:money: $150.00
Player Four:square 11:money: $100.00
Player Five:square 6:money: $50.00
Player Six:square 10:money: $75.00
```

5 DataGridView Columns' Setting

The **Colour, Money** and **Winner** columns of the **DataGridView** control are to be read-only. The **Name** column values can be changed by the user if they type in a string to replace the default name at any time.

6 The End Game

At the end of any round you need to check if the game has finished.

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 when any player reaches the **Finish** square.

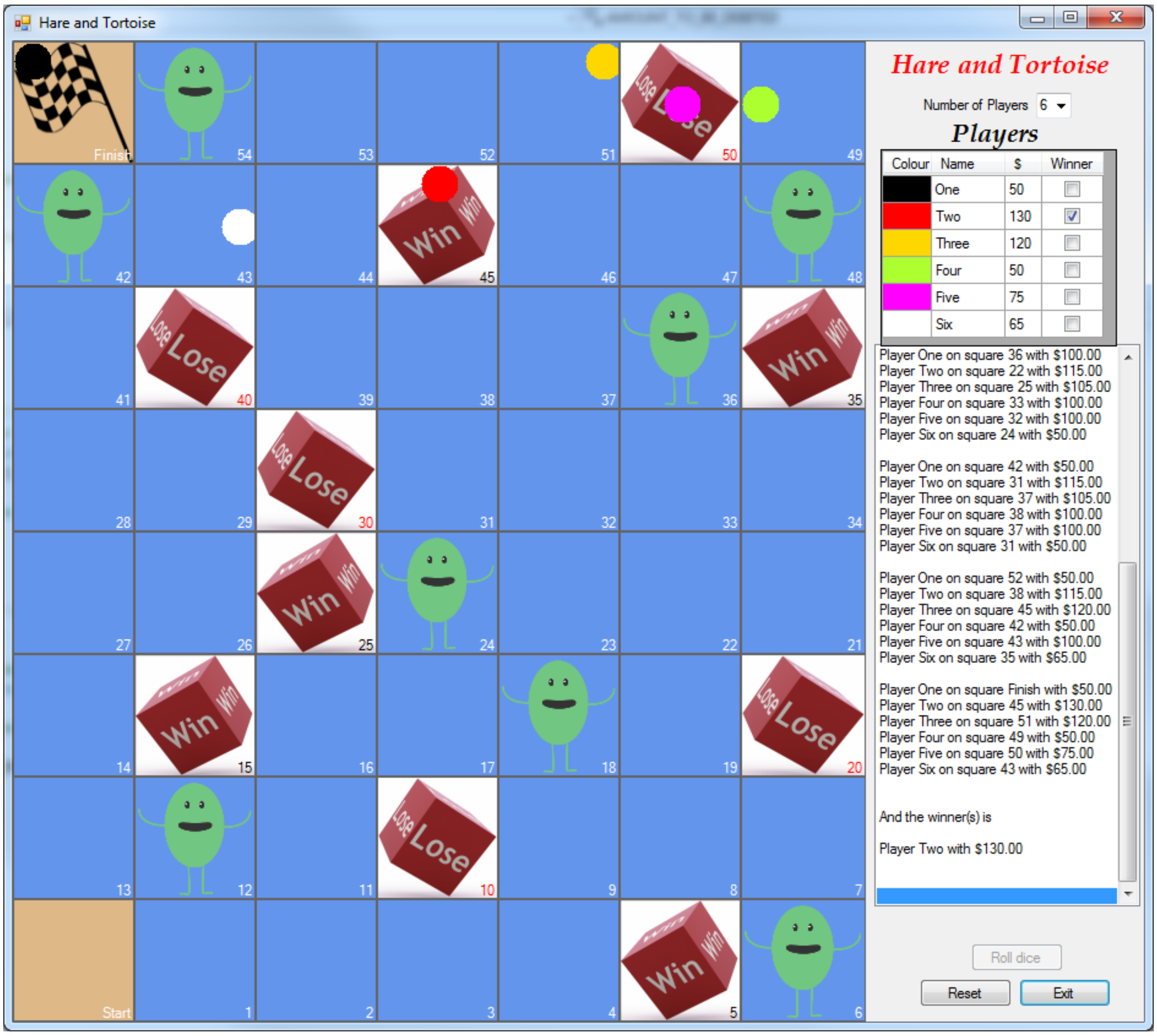
The winner will be the player who has the most amount of money at that time, even if that particular player has not yet reached the **Finish** square.

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.

Set the **hasWon** instance variable to true for the player(s) who has won the game. The corresponding **Winner** check box will be checked in the **DataGridView** control.

In the following screenshot, Player One has reached the **Finish** square; however Player Two is declared the winner with $130.



Hare and Tortoise

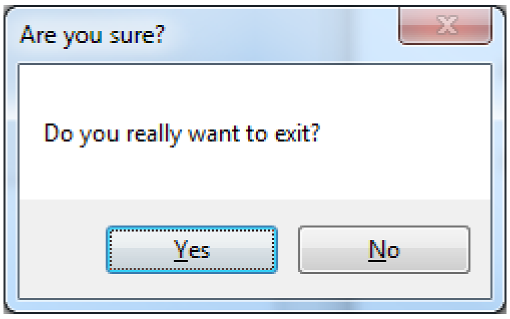
Players

Colour	Name	$	Winner
⬛	One	50	☐
🟥	Two	130	☑
🟨	Three	120	☐
🟩	Four	50	☐
🟪	Five	75	☐
⬜	Six	65	☐

```
Player One on square 36 with $100.00
Player Two on square 22 with $115.00
Player Three on square 25 with $105.00
Player Four on square 33 with $100.00
Player Five on square 32 with $100.00
Player Six on square 24 with $50.00

Player One on square 42 with $50.00
Player Two on square 31 with $115.00
Player Three on square 37 with $105.00
Player Four on square 38 with $100.00
Player Five on square 37 with $100.00
Player Six on square 31 with $50.00

Player One on square 52 with $50.00
Player Two on square 38 with $115.00
Player Three on square 45 with $120.00
Player Four on square 42 with $50.00
Player Five on square 43 with $100.00
Player Six on square 35 with $65.00

Player One on square Finish with $50.00
Player Two on square 45 with $130.00
Player Three on square 51 with $120.00
Player Four on square 49 with $50.00
Player Five on square 50 with $75.00
Player Six on square 43 with $65.00


And the winner(s) is

Player Two with $130.00
```

7 Adding Event Handlers

The **Exit** button event handler will first check that the user really wants to exit the game using a MessageBox with Yes, No buttons, similar to the following image.


Are you sure?

Do you really want to exit?

MessageBox.Show has 21 overloaded methods. Use the simplest which contains **MessageBoxButtons** as one of its parameter. You may have to read about **DialogResult** type for your event handler to work correctly.
The **Reset** button will start a new game with the player's token on the **Start** square and the **DataGridView** control showing the initial state of the game. That is, each player will have $100 and no one will be shown as the Winner. The number of tokens displayed will be the number visible in the **ComboBox** at that time.

When the game is first started, the "Number of Players" **ComboBox** will show 6. This has been chosen as it is simpler to construct six Player objects initially and then not use some of them if the user selects a smaller number of players.

The **ComboBox** class has a method **FindString(string)** which returns the index of the string in the **ComboBox** Collection of **Items.** This returned value can be used to set the **SelectedIndex** property of the **ComboBox.**

If the user selects to play with a smaller number of players then only that number of tokens will be on the **Start** square though the **DataGridView** control will still show all six players (to keep it simple). During the game only the correct number of players' information on the **DataGridView** control is updated as the game proceeds.

Once the **Roll dice** button has been clicked, the **ComboBox** is disabled until the end of the current game or the **Reset** button has been clicked. The **Roll dice** button should be disabled immediately after it has been clicked and not enabled until all the tokens have been moved.

The **Reset** button is not active during a round of play. But can be clicked once a round has finished and the game is immediately reset, whether the current game is finished or not. That is, the current players' tokens are moved to the **Start** square, the information in the **DataGridView** is reset. However the current value showing in the **ComboBox** is not changed but the **ComboBox** is enabled to allow the user to select a different number of players.