Class Assignment – Hare and Tortoise

Due Date: 28 October, 2014 – No late submissions accepted. Weighting: 40%

Specification Version 1.0 (25 September 2014)

*In this assignment you have the option of working in pairs. If you work alone, you will not receive any special consideration. Programming pairs need to be registered with Mike Roggenkamp by email before COB 17 October, 2014*

*Given more than 3 weeks to complete this assignment, there will be no extensions granted under any circumstances. If illness or other exceptional circumstances prevent you from submitting your assignment by the due date, email Mike,* [*m.roggenkamp@qut.edu.au*](mailto:m.roggenkamp@qut.edu.au)*;* ***Do not submit a formal request for an extension as the Examination period commences on 29th October and the exam for this unit is on 1st November.***

***Plan to submit by 26th October, 2014.***

**Table of Contents**

1. Introduction 1

2. The Task 1

3. The Hare and Tortoise Game 2

4. The Previous Programmer’s Notes 3

5. Finer details about the Hare and Tortoise Game board 3

6. Where to Start 6

7. Final Comment 8

8. Submission Details 8

# Introduction

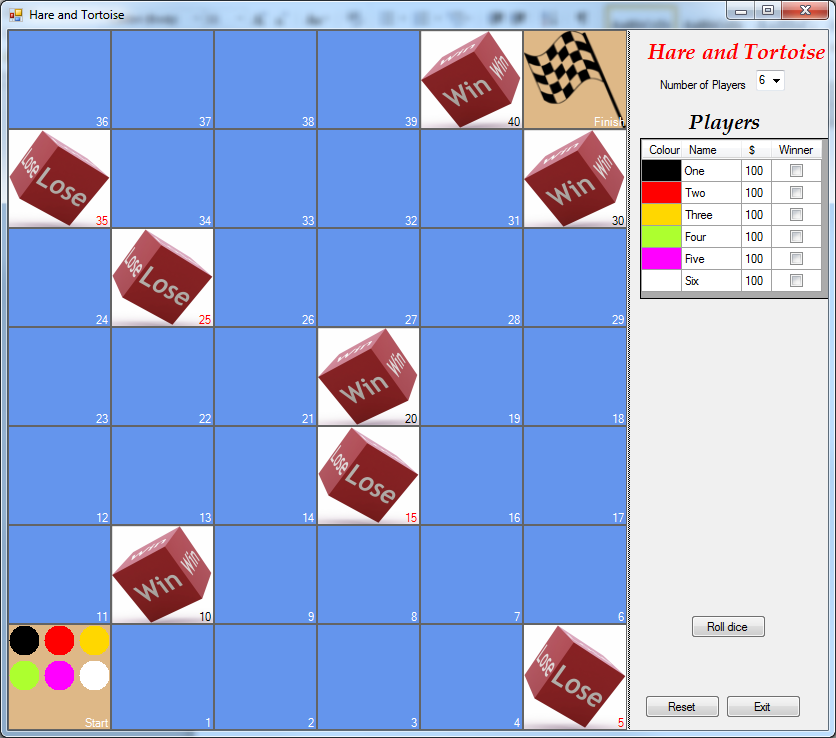
This assignment aims to give you a “real world experience” that occurs far too often in the workplace. You have been hired to complete as much as possible a project that has not been fully specified at this stage. You have been given (in your mind) an impossible delivery date. There is some supporting documentation to the project in addition to this document. In order to fulfil your contract you must deliver the prototype project which fulfils the stated functionality by the required date.

# The Task

The Good Product Software Company has hired you to complete the implementation of a prototype that is at an early stage of development. It is a Board game, named **Hare and Tortoise**. The programmer who was to implement the prototype has left the company at short notice and has left behind only the source code of various Classes and some notes regarding the prototype.

You are to develop a **Windows Form** version of the game. You are required to

1. Use **Windows Forms**, rather than other GUI technologies such as WPF, XNA, web-pages, game engines, etc. To keep this program relatively simple, do not use advanced techniques such as MVC or layered architectures. (If you don’t know what some of these acronyms mean, that’s fine. Just ignore them.)
2. Develop a GUI layout that is very similar to the screenshot below.



The above screenshot shows the game with 6 players, at the start of the game. Each of the 6 players has a coloured token (or “piece”) on the Start square. (If you want to see a bigger picture, right-click on the screenshot, select **Format Picture...,** then the **Size** tab, and then increase the **Height** – or **Width** – to 80% or more.).

# The Hare and Tortoise Game

This is entirely fictitious board game and bears no resemblance to the fable of the Hare and the Tortoise (by Aesop) or any exsting on-line game of the same name.

The game is played with a board consisting of 40 actual squares, a pair of six-sided dice and can be played by 2 or more players up to a limit of 6 (for the purpose of this assignment only). The squares are numbered 1 to 40.

The game commences with all players starting off the board. (Think of the Start as being square 0.) One at a time, the players take a turn. This involves rolling the pair of dice and advancing their respective tokens on the board the required number of squares. When all the players are done with their turn, it is called *a round*.

Each player has a sum of money, they all start with $100.

There are three types of squares on the board: “ordinary”, “bad investment lose’ and “lottery win”.

Squares 5, 15, 25 and 35 are “Bad Investment lose” squares. The effect when a player’s token lands on one of them is to subtract $25 from the player’s current sum of money, if the player has less than $25 his sum is set to zero. The sum cannot go below zero.

Squares 10, 20, 30 and 40 are “Lottery Win” squares. The effect when a player’s token lands on one of them is to add $10 to that player’s sum of money. As well the player continues his turn, rolls the dice again. Note: landing on the 40th square the player receives the additional money and the game is finished witout rolling the dice again.

The game continues until a player moves beyond the 40th square, reaching the Finish. (Think of the Finish as being square 41.) Players cannot move beyond the Finish.

When one or more players have finished, the winner will be the player who has the most money at this point of time, even if that particular player hasn't reached the Finish. Note that there may be more than one player with the highest amount of money, in this case they are all winners.

# The Previous Programmer’s Notes

The following text is the introduction to the previous programmer’s notes about the prototype and its classes. See the document, **My Working Notes,** for the full set of notes

*In the* ***Solution Explorer*** *window, the current VS solution contains two “projects” (or “folders”):*

* *The* ***GuiGame*** *project contains incomplete code for running the GUI version of the game – the classes in this project are described below. This project contains* ***GuiMain.cs*** *which contains the method* ***Main.***

* *The* ***SharedGameClasses*** *project contains the code which performs the logic of the game. Because of that, these classes should not do any I/O at all.*

# Finer details about the Hare and Tortoise Game board

The screenshot on the page 2 illustrate the basic layout of the GUI, there are some finer details that the screenshot does not show, or does not show clearly. Do not start implementing the GUI until you have finished reading this document and the programmer’s notes concerning the ***HareAndTortoiseForm class*** in **My Working Notes.**

Position the GUI Controls as shown. While you are not expected to get their locations and sizes correct to the very last pixel, your GUI layout should look fairly close to the one given in this specification. No additional dialog boxes or message boxes are to be used (unless you need them to assist you with debugging though they are not to appear in your submitted assignment when it is executed).

1. The game board is displayed by using a TableLayoutPanel. While you will need to tell this TableLayoutPanel how many rows & columns you need – don’t waste your time adding a Control for each of the 42 (7 x 6) squares on the board, in the **Design View**. It is much easier to do this in your program code later. This will be explained in the programmer’s notes.
2. The fonts used for labels, etc., are just the normal fonts that the **Windows Forms Designer** uses, except in the cases of the two larger labels – in the upper right-hand part of the form – which both use a 16pt, bold, italic **Book Antiqua** font.
3. The Start and Finish squares have **Color.BurlyWood** as their background colour (BackColor property).
4. The **Win** and **Lose** images shown on the board above are included as part of the prototype given to you as well as the checked flag image on the **Finish** square You are not expected to create these images, although you will have to write some code to make them appear on the board (in **SquareControl.cs**).
5. The numbers shown in each square are usually white, but the **Bad Investment** squares (**Lose**) are shown in red, and the **Lottery Win** squares (**Win**) are shown in black.
6. The **Number of Players** ComboBox – upper right-hand corner – allows the user to choose a value between 2 and 6. When using the **Windows Forms Designer**, you should enter the allowed values: 2 through 6 directly, rather than writing program code to insert these values.
7. When the game is first started, the **Number of Players** ComboBox always shows 6, the maximum number of players. This value has been chosen to make things a little bit easier for you to program. It is simpler to always construct 6 Player objects when the game starts, and then not use some of them – when the user selects a smaller number of players – rather than having to dynamically create/delete Player objects. By the way, the Player objects are held in the **Players BindingList,** in **HareAndTortoiseGame.cs**.
8. Whenever the user selects a different number of players in the ComboBox, the game is immediately reset, whether the current game is finished or not. This means that the user cannot change the number of players in the middle of a game and then continue the same game. When the game is reset like this, the Start square shows the correct number of tokens for the newly-selected number of players. E.g. if the value “2” is selected, then only two tokens are shown.
9. In the **Players DataGridView** – upper right-hand side – only values in the **Name** column can be changed, by the user(s) typing in their names to replace the default names. The **Colour, Money and Winner** columns are read-only. They cannot be changed by the user(s) entering/selecting new values.
10. The overall purpose of the three buttons in the right-hand lower corner should be obvious. Each of the buttons can be clicked at any stage of the game, although the **Roll dice** button and **Reset** button should be disabled during a round of play and whenever a game is finished, and re-enabled at the start of the next game.

Clicking the **Reset** button moves all the players’ tokens back to the **Start**, resets the information shown in the **Players DataGridView**. It does not change the value selected in the **Number of Players** **ComboBox**.

1. The user can resize the game by dragging on its borders. (Don’t worry if doing this causes some of the square numbers to display strangely.) To support this, most controls should have the default set of values for their **Anchor** property, which is:

**Anchor = Top, Left**

So that the three buttons stay in the bottom right corner, each should have:

**Anchor = Bottom, Right**

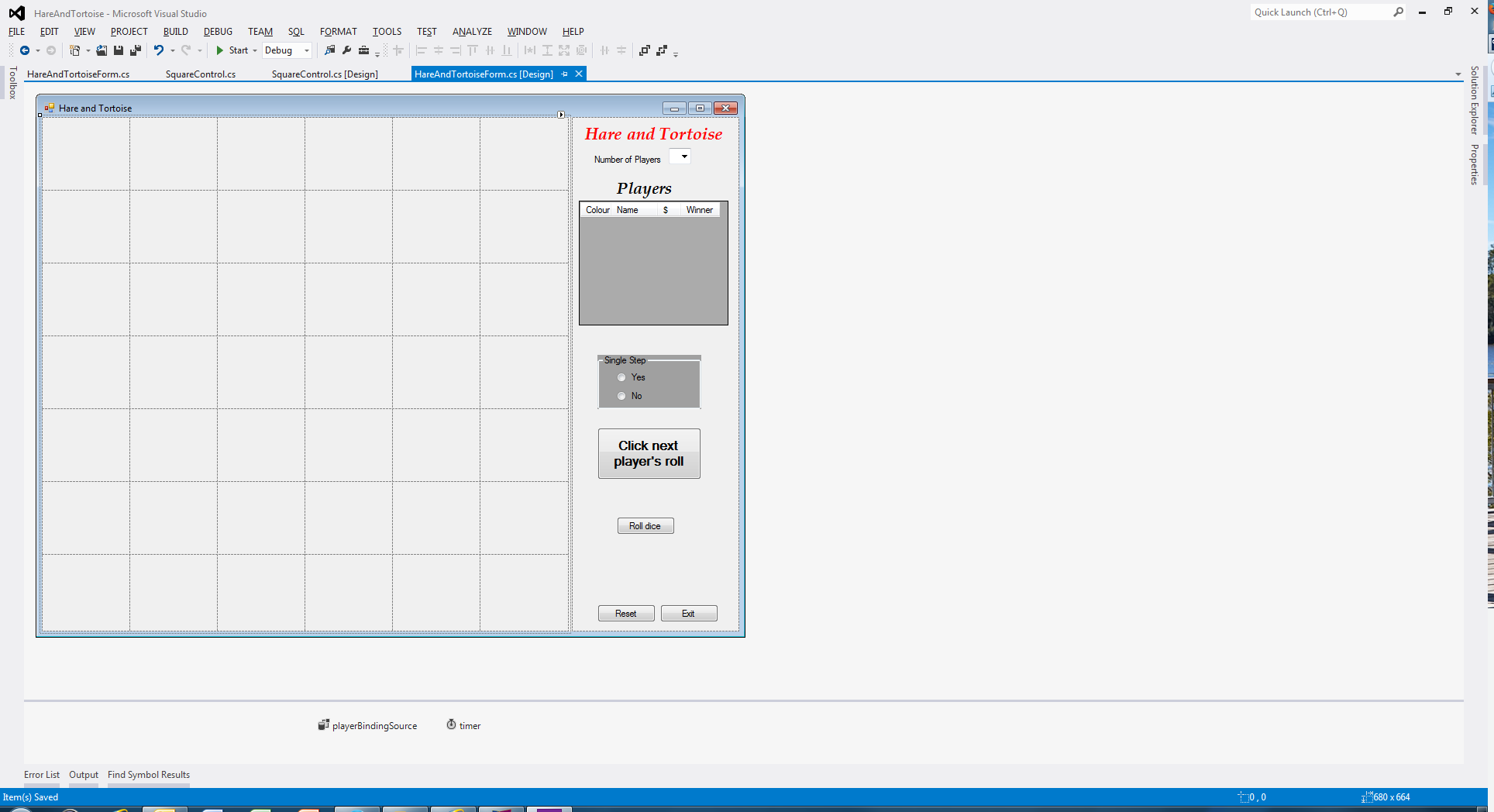
1. There is a **Group Box,** labelled **Single Step,** which contains two radio buttons, **Yes** and **No.** As well there is button a buttonwith the text **“Click next player’s roll”.** These controls are for an extension to the assignment which will be described later. These controls are not visible in your initial implementation of the GUI as can been seen from the screenshot on page 2.

# Where to Start

Use **Design View** to add controls to the prototype **HareAndTortoiseForm**, *so that it has the overall layout as shown on page 2.* Don’t expect to get your form looking exactly the same as the screenshots, to start with. Some parts – the board’s squares and the Players DataGridView – will initially look quite different.

As well as reading what the previous programmer said about this form, you should refer to Lectures X & XX and the related Practicals, for information on using **Design View**.

As stated in previous section 5(a), do not waste your time by adding controls for each of the 42 squares in the board. When you have finished creating the board’s **TablePanelLayout**, you should see a 7 x 6 grid (in **Design View** only), but not the individual coloured squares. They will come later when you start writing code to initialise the board. Your GUI layout should now look like the following screenshot.



Once you’ve got all (or almost all) the controls added to the form – even though some of them appear blank initially, or show the wrong columns – it’s time to start adding some code to make those controls work a bit better. Most of this work will require you to add code to that already provided in **HareAndTortoiseForm.cs**, so start by trying to make sense of what is already there. You probably won’t understand it all to start with, also read the previous programmer’s notes, **My Working Notes**. There are also comments in the code that should help explain the purpose of the methods provided and in some cases algorithms to assist you to implement the functionality of the method. The following section should eb read in conjunction with the description in **My Working Notes**.

**Adding Code to the Game controls**

While some parts of the code behind the GUI can be implemented in any order you like, I recommend doing things in the following sequence. It is easier to do most of the steps by assuming there are always 6 players, and allowing for a different number once you are able to play the game.

1. Get the board to display all the squares, by adding code to the **SetupGameBoard** method in **HareAndTortoiseForm**.. To do this, you must construct a **SquareControl** object for each of the squares in the board, including the **Start** and **Finish** squares. Don’t try to get all the squares displayed perfectly correct at your first attempt. Getting a board full of blue squares is a good start. You will eventually need to call the method **MapSquareNumToScreenRowAndColumn** to map an individual square’s number to the correct position in the **TableLayoutPanel.**

**MapSquareNumToScreenRowAndColumn** needs to be implemented. The top left corner of the **TableLayoutPanel** is position (0, 0) and the bottom right corner is position (6, 5). This method can be implemented after you can display 42 blue squares.

1. Enhance your code so that the **Start** and **Finish** squares have the right colour, and that the square numbers are in the correct sequence – they should go right-to-left on some rows. You don’t need to see the player tokens yet.
2. Modify the code in **SquareControl.cs** so that the **Win** and **Lose** images are shown on the board on the correct squares, and that those squares have red number or black numbers as shown in the screenshots. Also add the **Checkered Flag** image to the **Finish** square. (If you get stuck on any of this, go on to the next step. We’re just making the board look fancier, this can be done later)
3. Create an event-handler for the **Roll dice** button, which calls a method (which you have to yet to write) in the **HareAndToiseGame** class to “play one round” of the game. The “play one round” method will call the **Play** method of Player class for each player. As yet you don’t have player-tokens displayed on the board, so you should consider placing break points in the “play one round” method and the Play method to see that the code is working after you have written the necessary code.
4. If you haven’t already done so, use **Design View** to create the **Players DataGridView**. (Then add one line of code to your **SetupPlayersDataGridView** method so that the **playerBindingSource** knows that its **DataSource** is the **hareAndTortoiseGame.Players** **BindingList**. By using **Design View** to set the properties of this **DataGridView**, you avoid the need to write much code at all. This is one of the reasons that the **DataGridView** is a very useful way of displaying data.

(If you have any problems with this step, you can still do the steps below.)

1. Add code to the **ResetGame** method so that all 6 player-tokens appear on the **Start** square, when the game is first opened, and whenever the game is reset. You code should make use of the **UpdatePlayersGuiLocations** method. You have to write the code inside that method too, but once you’ve done that you’ll have a useful method for your **ResetGame** method and for some other methods too.
2. Create an event-handler for the **Reset** button.
3. Modify your **Roll dice** button’s event-handler, so that all the players’ token are moved when you click that button. Because of your existing methods, this should be fairly easy.
4. Add/change code so that the user-selected value in **Number of Players** ComboBox works as expected. In particular, only the correct number of player-tokens should appear on the board. As shown in the earlier screenshots of the game with two players, all six players are still shown in the Players DataGridView (for simplicity), but only the correct number of players are updated as the game proceeds.
5. Now make the **GroupBox** containing the radio buttons visible at the start of a Game and make the **Roll Dice** button not enabled. So that at the start of a Game the user decides whether to see each token move separately each requiring the **Roll Dice** button to be clicked or all together on the one click of the **Roll Dice** button.
6. The previous step may require you to add additional methods to the **HareandTortoiseGame class** or substantial redesign one or more existing methods in order to get your code working correctly in either single step mode or all at once. .
7. This last step is for those wishing to obtain a 7 or high 6 for this assignment. Implement a simple form of animation that shows each player *moving one square at a time*. E.g. if Player One is on the Start square and rolls 8 on the dice, then they are displayed as moving to square 1, then to square 2, and so on, until they reach square 8, rather than moving from the Start to square 8 in one go. Player 2 then moves similarly, etc. This animation is something that you should only work on once you have the rest of the assignment working..

The challenge in this assignment is to complete what you can before the due date. Better to hand in a working project which does something correctly than one which does not run.

# Final Comment

Though all care has been taken in the production of this specification and related documentation, there may be a need to notify by class email any alterations/clarifications to this specification and related documentation

**SO CHECK YOUR QUT EMAIL DAILY.**

**Working in Optional Pairs**

Ensure that both people in the group are involved and are responsible for doing some part of the assignment. Do not forget to register your group details via email to Mike before COB on 17 October, 2014. Include the full name and student number of each person in your email as well as a Group name.

# Submission Details

Will be provided by 17 October 2014 on Blackboard.