**Conestoga College - ACS & IT**

**Programming Microsoft Web Technologies – PROG2230**

**Assignment #3: Press Your Luck!**

**Due: Friday, Nov 12th, 2021 (by Midnight)**

**Total marks: 40 Worth: 6%**

### Introduction

~~The goal of this assignment is to create a simple betting game that uses sessions, serialization and cookies to keep track of most of the information in this game. A lot of the information can be found in Chapter 9 of the class text, though it does use concepts from other chapters to help you practice.~~

### Please note:

~~Your solution~~ ***~~must only use techniques covered in class~~***~~. Also, I remind you to make sure you do your own work on this assignment and resist any urge to copy code from any other source - e.g. your classmates, the web, etc. Not only is this the only way to learn how to program but also everyone’s solution will be run through~~ [~~Moss~~](https://theory.stanford.edu/~aiken/moss/) ~~to check for academic integrity violations. There is zero-tolerance for such violations and any encountered with be dealt with in accordance with~~ [~~Conestoga’s policy~~](https://lib.conestogac.on.ca/academic-integrity/penalties).

### Project Name

As discussed in class, please name your Project AS3\_<Initials><Last 4 digits of student number>. (Eg. AS3\_RL0000)

### Hints

The general idea is that there will be time to work on these questions in class and, if necessary, I can offer hints if I see that you are struggling with certain parts. Also, you should try to pace out your work on this assignment over the coming weeks.

### How will it be graded?

Accompanying each assignment will be an Excel marking sheet that details how your grade will be calculated so you obviously want to make sure that you are doing everything as it’s laid out there.

### What/how to submit?

Zip up your entire solution into ***one zip file*** and submit that file to the eConestoga dropbox for the assignment. You can submit multiple solutions but only the latest (i.e. most recent) one will be looked at and evaluated.

### What You’re Building

Your goal is to build a simple betting game. To start with, you enter your Name and the number of coins you’re starting with.

To start, you choose how many coins to bet, then 12 tiles are shuffled face down.

Each round you either choose a tile or take the coins you’ve won so far. If you choose a tile and it shows 0, you bust and lose your bet. If it has a number, your bet is multiplied by that number and the remaining tile values are doubled.

Once you choose to take the coins, the amount is added to your total coins, and you can start a new game or cash out and stop playing.

In addition to this, you’ll also create audit records to keep track of winnings and losings.

### Highlights

The following highlights some of the techniques you will be employing in this assignment. Detailed instructions are included below.

* The current player name and their total number of current coins will be stored in a cookie so that it can persist across sessions
* The current game state (for the current player) will be stored in a Session
* Essentially a game consists of a List of (randomly generated) Tile objects. You will use the NewtonSoft.Json package to serialize/deserialize the current game to/from the session as Json strings (as done in chapter 9).
* You will implement some static helper methods along the way to make it easier to interact with the required Session & Cookie data
* Once that is all working you will build on the solution to store an audit trail in a SQL server database

### Tour the Code

To start, download the code called “Assignment 3 – Starting Code”, load it and make sure it compiles for you.

Run the project and you’ll see a link on the default page that will take you to a list of tiles.

These should all be face up, (they will also be huge because we haven’t used bootstrap to make the page look pretty yet) You’ll see red “X” tiles for any that are 0, and green “$” for any that have an amount. Currently, tiles will always display the amount, but we’ll be changing that.

This assignment makes use of static helper classes and two have been created for you. Classes like this can be used to create code that you may call many times in your application. This makes the code easier to maintain as well as easier to write. Alternatively, if you prefer, you could encapsulate session access inside a class, much like the authors do in chapter 9.

Go to the class, Helpers/GameHelper and you’ll see there is a single method called “*GenerateNewGame*”. This method generates a list of 12 tiles, with values randomly assigned to it. Change the line that says “Visible = true;” to “Visible = false;” and re-run the code.

Text

Description automatically generated

You should now notice that the tiles are all face down. From this point on, you should not need to make any changes to the *GenerateNewGame* helper method.

I’ve also created a *CoinHelper* and have added hints via comments, as well as some starting code, in each class about the various types of helper methods you may find useful to create. This is optional, but it will make your life a lot easier.

### Controllers

My expectation is that, by the end of this assignment you’ll have four controllers. Empty controllers have been created for you.

1. *HomeController* – The default landing page as well as the page that lets you enter your bet
2. *PlayerController* – The controller that create and cashes out player data.
3. *GameController* – The main controller of the project that communicates game state to and from the client
4. *AuditController* – This controller will display a list of audit records.

### Work Breakdown

In order to build more complicated web sites, it’s often helpful to break the work into manageable steps.

**I’ve broken down the work for you into 8 parts, and I strongly recommend getting each part working before moving on to the next.**

In each section, you’ll find screen shots that you can use to work from.

1. Create a new player and let them cash out.
2. Have the player enter a bet, then start a new game.
3. Display the initial state of the game to the player
4. Implement the game rules when a player chooses a tile
5. Display helpful messages to the player.
6. Create your audit database.
7. Add in audit records and create an audit list.
8. Create a filter for the audit list that persists in session.

### Part 1) Create a new player and let them cash out.

We’ll be keeping track of two pieces of data for the player; their name (player-name), and the total amount of coins they have (total-coins). Because we want this data to persist across sessions, we’ll be using cookies to store and retrieve this information

1. Install any Nuget packages and make any coding changes needed to enable Sessions and Serialization. (See Chapter 9 if you’ve forgotten)
2. Whenever we try to get to the default landing (i.e., Home) page, if the player-name cookie is empty, force the use to go the player controller in order to enter in new information
3. From the player controller:
   1. Create a Player object that keeps track of the player’s name and the starting number of coins. Name is a required fields and the total number of coins should be between 1.00 and 10,000.00
   2. Create a view that collects this information, and, if it passes validation, stores it into cookies, then passes controller back to the Home/Index page
4. Modify the common navigation bar to display the total number of coins the player has along with a button to cash out.
   1. If they click on “Cash Out”, clear the cookies you’ve created and send the user back to the screen to enter a new player

Graphical user interface, text, application, email

Description automatically generated

### Part 2) Welcome the player, collect their bet, then start the game

1. Remove the direct link to the tile list from the homepage. It’s no longer needed.
2. On the home page, assuming that the player and total coins is stored in a cookie, welcome the player by name then collect their bet and send them off to the Game controller to start a new game.
   1. The amount of the bet must be greater than 0
      1. **Note:** Creating custom validation won’t be covered until later in the course, so we won’t check to make sure that the player has enough coins to bet with.
   2. We’ll be saving all information about a single game into session, so once the user has entered a bet, we’ll need to deduct the betting amount from their total coins, then save the amount they bet into a session variable.
      1. **HINT:** Since the amount they can bet and win is a double, you’ll need to store the value as a string and convert it back to a double when needed. Also, this may be a good time to consider creating helper functions to keep track of coins.

Graphical user interface, application

Description automatically generated

### Part 3) Display the initial state of the game to the user.

1. We’ll be using serialization to store the list of tiles create into session.
   1. If the list of tiles isn’t in session, generate a new game and store it in session, otherwise, deserialize the information that’s there into a List of Tile.
2. Create a View Model for the page that contains the list of tiles along with the current betting amount
3. Make a number of UX changes to the page
   1. First, use the bootstrap grid system to organize the tiles, so that 6 tiles appear on each row
   2. Above the tiles, display the users bet along with a link either says “TAKE THE COINS!” if the current bet is above 0, and “Try Again” if the bet is 0 (We’ll add functionality later)
      1. To make things look nice, feel free to use bootstrap to make this look like a button
   3. Where the value is displaying, if the tile is not yet visible, instead create a link that says “CHOOSE” beneath the image. Keep the value if the tile is visible.

Qr code

Description automatically generated

### Part 4) Implement the game rules when a user chooses a tile.

1. When a user chooses a tile we call out to the Game controller Reveal action. (**Hint:** each title has a TileIndex to keep track of each tile, which you should pass in) There, set the tile to visible, and then one of two things should happen:
   1. If the value is 0, the player has lost. Set the current bet to 0, then reveal all tiles so that player can see what they could have won
      1. If the user still has total coins, clicking on Try Again should take them to enter a new bet.
      2. If the user now has 0 total coins, force the user to cash out and prompt for a new user.

A picture containing application

Description automatically generated

* 1. If the value is not zero, multiple the current bet by that amount, double any remaining amounts that have not yet been chosen, then let the user pick another tile (or TAKE THE COINS)

Qr code

Description automatically generated

1. If a user selects takes the coins:
   1. Add the amount of the current bet to their total coins, then take the user back to make another wager.

### Part 5) Display helpful messages to the player

Now that we have the game working, let’s add a few helpful messages to the program.

1. Using TempData, display messages to the user when the following event happen:
   1. If the user picks a 0 tile: “On no! You busted out. Better luck next time!”
   2. If the user pick a non-0 tile: “Congrats you’ve found a XXX multipler! All remaining values have doubled. Will you Press Your Luck?”
   3. If the user takes the coins: “BIG WINNER! You chased out for XXX coins! Care to press your luck again?” (**Hint:** This message will be displayed on the page where the player can enter their next bet)
   4. If the user cashes out completely: “You cashed out for XXX coins”
   5. If the user’s total coins ever drops to 0: “You’ve lost all your coins and must enter more to keep playing”

Graphical user interface, application

Description automatically generated

### Part 6) Create your audit database

Now we’ll want to keep track of audit records when various events happen. Note that this project has not been set up to create a database.

1. Make changes to the project so that you can create a Entity Framework Code First database. Review Chapter 4 (Week 2) if you’ve forgotten how to do this.
2. For this project you’ll create a new database called “PressYourLuck\_<student-alias><last-4-digits-of-student-number> (eg. “PressYourLuck\_rlegood1234)
3. You’ll create two Models to use as tables.
   1. AuditType
      1. This will have two required fields, AuditTypeId and Name
      2. Seed the database with four audit types:
         1. Cash In
         2. Cash Out
         3. Win
         4. Lose
   2. Audit
      1. This will have four fields; AuditId, PlayerName, CreatedDate and Amount
      2. …as well as a foreign key relationship to the AuditType model.
      3. CreatedDate is the date and time that the record was created. **Hint:** Use DateTime.Now to get the current date and time.
4. Create your migration and run update-database to create the database

Text

Description automatically generated

### Part 7) Add in audit records and create an audit list.

1. Each time one of the following occurs in your program, create an audit record and save it to the database:
   1. Cash In – Whenever a new player signs up. The Amount is the number of coins they start with
   2. Cash Out – Whenever a player stops playing and cashes out. The Amount is the number of coins they ended with
   3. Win – Each time a player wins a round, record how much they won.
   4. Lose – If a player loses, record what their original bet was
      1. **Hint:** You may need to create a new session variable to keep track of the amount of the original bet
2. In your AuditController, display a list of audit records, stored in reverse chronological order.
   1. Update the navigation bar to create a link to this page.

A screenshot of a computer

Description automatically generated with medium confidence

### Part 8) Create a filter for the audit list that persists in session.

1. Create a tab bar, above the audit record table that lists all Audit Types and also an “All” option.
   1. If a tab is selected, filter the table to only show records of that type.
   2. Use a session variable to remember which AuditType was selected. This session variable should persist for the entire session. (ie. If I leave the page and come back, the system remembers the Audit Type that was chosen)
   3. **Hint:** You can use the bootstrap “active” class to show which tab is active

Table

Description automatically generated