**CIS 162 Project 3**

**A Game of Pig**

**Due Date**

* at the start of lab on

**Before Starting the Project**

* Read chapters 5.1 - 5.4 and 6.1 – 6.3
* Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

* *use* a Graphical User Interface (GUI)
* write methods to meet specific requirements
* *write* conditional statements with boolean expressions
* *write* a looping construct

**Game Rules**

* The game requires two six-sided dice.
* Players take turns until someone wins by earning at least 100 points
* Players begin a turn by rolling both dice and tallying the points. The current player continues rolling and accumulating points until rolling a '1' or choosing to 'hold'. If the player rolls a 1 then all points from this round are lost. If the player rolls a pair of 1s then all player points are lost (set to zero).
* Play moves to the next player
* Your game will be between a person and the computer. The person always goes first.

**Step 1: Create a New BlueJ Project**

**Step 2: Add the provided GVdie class to your project**

Rather than writing your own Die class, we are providing a completed class for you. There are two ways to add this file to your BlueJ project.

Option 1: Create a new class in BlueJ called GVdie and delete all of the provided code. Cut and paste the provided GVdie.java into the newly created class. It should compile with no errors.

Option 2: Or, right click on the GVdie.java file to download on your computer in the folder for the newly created BlueJ project. Pay attention where it is saved! Within your BlueJ project, select Edit 🡪 Add class from file…

Do not make any changes to the code. Instead, read the internal documentation to understand how to use it. The most important methods are:

* void roll ( ) – randomly change the value of the die (1 - 6).
* int getValue ( ) - returns the current value.

**Step 3: Create a class called PigGame (55 %)**

Create a class, called *PigGame*, with the following instance fields and methods. Do not create additional methods or make any changes to the following requirements without approval from your instructor.

**Class Fields**

* Provide appropriate names and data types for each of the instance variables. Maintain two GVdie objects, integers for the player’s score and another for computer’s score, an integer for the current round score and a final int for the winning score of 100.

**Constructor**

This special method has the same name as the Class and generally initializes the fields.

* public PigGame( ) - a constructor that initializes all of the instance variables to appropriate values, *instantiates* two dice and prints a welcome message.

myDie = new GVdie();

**Accessor Methods**

An accessor method should not modify class fields.

* public int getRoundScore ( ) – return the current round score.
* public int getPlayerScore ( ) – return the player’s score.
* public int getComputerScore ( ) – return the computer’s score.
* public boolean playerWon ( ) - return true if the player score is currently high enough to win. Otherwise, return false.
* public boolean computerWon ( ) - return true if the computer score is currently high enough to win. Otherwise, return false.

**Mutator Methods**

A mutator method may modify class fields.

* private void rollDice ( ) - roll both dice once. If either value is '1' then set the round score to 0. Otherwise, increment the round score by the new total. Print both dice values and the round score. This method is used by both the player and the computer. Note, it is a private method.
* public void playerRolls ( ) - performs the first half of the player turn: 1) invoke the rollDice( ) method, 2) print the player's score if the round is over, or 3) print an appropriate message if the player wins. Refer to the sample output below. Set the player score to zero if double 1s are rolled.
* public void playerHolds ( ) - performs the second half of the player's turn: 1) update the player's score, 2) reset the round score to zero and 3) print the player's score. Refer to the sample output below.
* public void computerTurn ( ) - performs the computer's entire turn: continue rolling the dice and updating the round score until: 1) a '1' is rolled or 2) the round score surpasses 19 or 3) the computer wins. Set the computer score to zero if double 1s are rolled. Use a while loop to repeatedly roll the dice. Update the computer's score and the round score after each. Print the computer's score. Refer to the sample output below.
* public void restart ( ) – reset all instance variables to start a new game. **Do not** instantiate new dice objects.

**Coding Style (10 %)**

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU [Java Style Guide](http://www.cis.gvsu.edu/java-coding-style-guide/).

**Step 4: Software Testing (15 %)**

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious. Another approach is to write methods that automatically play a game and therefore call all of the other methods. You can carefully review the results of a game to confirm that it works correctly.

* private void playerTurn ( ) - simulates the player's entire turn: continue rolling the dice and updating the round score until a '1' is rolled or the round score surpasses 19 or if the player wins. Update the player's score and the round score. Print the player's score. Keep this method relatively short by calling playerRolls() and playerHolds() where appropriate.
* public void autoGame ( ) - automates an entire game by alternating between a player's turn and computer's turn until one of them wins. Reset all values before starting a new game but not after. Keep track and report the total number of turns taken and who wins. See sample results below. Keep this method relatively short by calling playerTurn() and computerTurn() where appropriate

**Sample Text Output**  
The following sample shows partial results of a game. Your messages can be more creative as long as they convey the necessary information. Note: totals turns is only displayed during automated games.

Welcome to Amanda's Game of Pig!

4 1 Round Score: 0

---- Your score: 0

2 4 Round Score: 6

6 3 Round Score: 17

1 3 Round Score: 0

---- Computer score: 0

5 1 Round Score: 6

4 3 Round Score: 13

4 5 Round Score: 22

---- Your score: 22

*later in the game…*

6 2 Round Score: 8

4 4 Round Score: 16

3 5 Round Score: 24

---- Your score: 92

6 2 Round Score: 8

6 4 Round Score: 18

3 2 Round Score: 23

---- Computer score: 73

6 3 Round Score: 9

---- Your score: 101

You won!

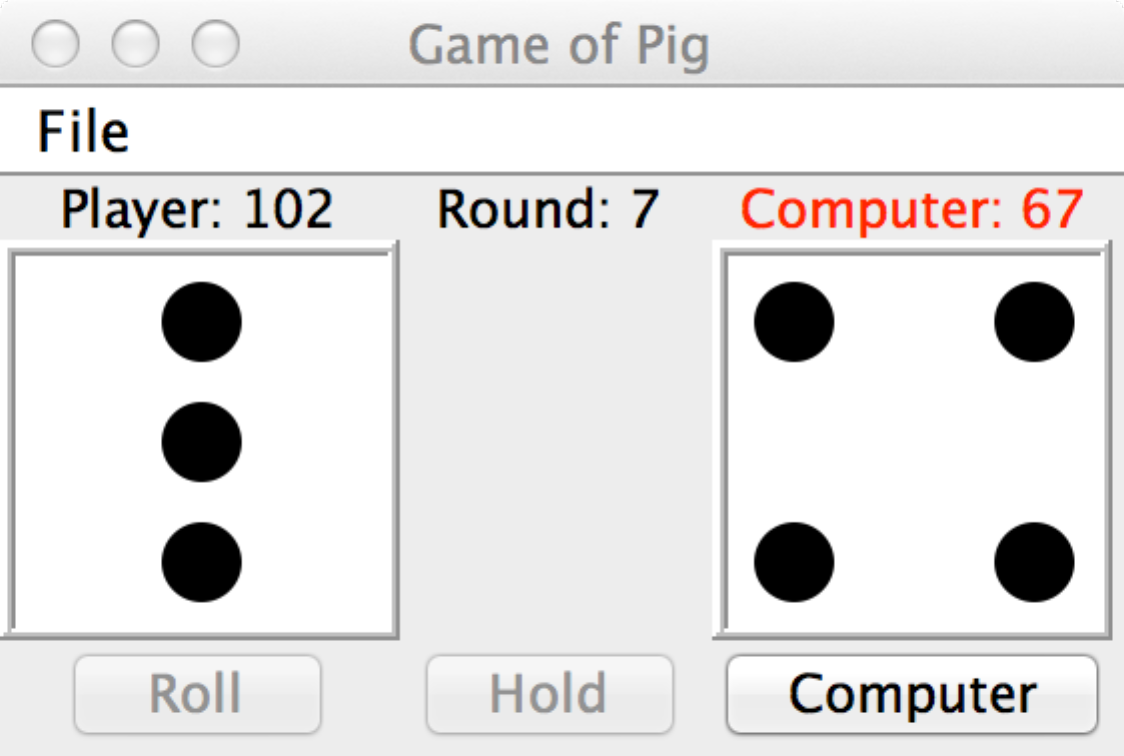
Total turns: 12

**Step 5: Enhance the PigGame class to support GUI (10 %)**

Now that you have the basic game working within it’s own class it is time to create a more interesting graphical user interface for the player to use. Add the following fields and methods to the PigGame class.

* Add a boolean instance variable to indicate if it is currently the player’s turn. You will need to make several changes throughout the PigGame class to set this variable to true when it is the player’s turn and false when it is the computer’s turn. Reminder, all games start with the player.
* public boolean isPlayerTurn ( ) - this one line method returns true if it is the player's turn or false if it is the computer's turn.
* public GVdie getDie (int num) - return the requested die. Legal values for the parameter are 1 or 2.

**Sample PigGUI Screenshot (see Step 6)**

****

**Step 6: Modify the provided PigGUI class (10 %)**

Rather than writing your own GUI class, we are providing an almost complete class to get you started. Create a new class in BlueJ called PigGUI and delete all of the provided code. Cut and paste the provided code from (PigGUI.java) into the newly created class. It should compile with no errors. If it doesn’t compile then it probably means you used different names for your methods than what was specified. Do not make any changes to PigGUI to make it compile.

We modeled our design on examples from the book in section 6.3. Read the code and comments carefully to understand how it works. You start the program by invoking the PigGUI’s main method. At this point, nothing will happen when the player clicks on the roll button. Look for FIX ME comments throughout the code for you to make improvements.

Make the following changes to the PigGUI class

* Change the JFrame title to include your name
* Add statements in the constructor to register each of the three buttons with the ActionListener. Read the internal comment that shows where to add the statements.
* A computer button is defined and created but has not been placed on the JPanel. Add the necessary statements to place the Computer button below the die on the right side.
* Add a statement to set the background color to your choice

setBackground(????);

Make the following changes to the actionPerformed() method. Read the internal comments for clues.

* Update the three labels with the current scores.

round.setText(“Round: “ + game.getRoundScore());

* Add if statements for each of the buttons and invoke the appropriate game method:

if (buttonPressed == roll)

game.playerRolls();

* Display a pop up message after either the player or computer wins. Use a JOptionPane.showMessageDialog as described in section 6.6.

if (game.playerWon()}{

// display message

}

**Grading Criteria**

There is a 50% penalty on programming projects if your solution does not compile.

* Stapled cover page with your name and signed pledge. (-5 pts if missing)

**Late Policy**

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

* The first 24 hours (-20 pts)
* Each subsequent weekday is an additional -10 pts
* Weekends are free days and the maximum late penalty is 50 pts.

**Turn In**

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

* Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
* Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the School of CIS Guidelines for Academic Honesty.
* Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, “I spent 7 hours on this project from January 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document.”
* Source code - a printout of your elegant source code for the PigGame and modified GUI classes. You do not need to provide the GVdie class.
* Demo – be prepared to demo your project on a lab computer or your laptop. I will ask you to perform a variety of tasks including play the game and invoke the autoPlay() method.