**CIS 162 Project 3**

**Farkle GUI**

**Due Date**

* See on Blackboard

**Before Starting the Project**

* Read chapter 9 (GUI).
* Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

* *create* a Graphical User Interface (GUI)
* *use* an ArrayList to store Objects

**Model-View Design Pattern**

A common design strategy is to divide applications into at least two parts: the user interface (View) and a back end that takes care of the data (Model). For this project, we provide a completed Model class that simulates a dice game, called Farkle, that you will eventually implement yourself in Project 4. For now, you do not need to know any of the game rules.

The Model-View pattern enforces clear responsibilities between the model and view. For example, the GUI (view) should not be responsible for maintaining any data or knowing how the data are processed. The sole responsibility of the GUI is to support user interaction with the Model.

**GUI Features**

The game supports multiple players using six dice.

* a player rolls all six dice to start and then selects one or more dice.
* a player then chooses to roll again or pass the dice. Only unselected dice are rolled.
* the next player is chosen by clicking a radio button
* players can display the best score by selecting File -> Best Score
* players can start a new game at any time by selecting File -> New Game
* players can quit at any time by selecting the File -> Quit

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

**Step 2: Use Existing GVdie**

Rather than writing your own Die class, we are providing a completed class for you. Create a new class in BlueJ called GVdie and delete all of the provided code. Copy and paste the provided code from (GVdie.java) into the newly created class. It should compile with no errors. Do not make any changes to this code but you are encouraged to read through the source code to see how it works.

**Step 3: Create a Player class**

Create a simple class that will be used in this project and more prominently in Project 4. The class maintains four pieces of information for a game player: name (String), score (int), subtotal (int) and number of turns taken (int).

**Constructor**

A *constructor* is a special method with the same name as the class and generally initializes the fields to appropriate starting values.

* public Player (String pName) – set the player’s name to parameter pName and the remaining values to zero.

**Setter and Getter Methods**

An *accessor* method does not modify class fields. The names for these methods, which simply return the current value of a field, often begin with the prefix ‘get’.

* public String getName() - returns player’s name.
* public void setName(String n) - set player’s name.
* public int getScore() - returns player’s score.
* public void setScore(int s) - set player’s score.
* public int getSubtotal() - returns player’s subtotal.
* public void setSubtotal(int s) - set player’s subtotal.
* public int getTurns() - returns player’s number of turns.
* public void setTurns(int t) - set player’s number of turns.

**Mutator Methods**

A mutator method performs tasks that may modify class fields. The following methods will make more sense in Project 4 but simply follow the specifications closely for now.

* public void addToSubtotal(int amt) – increase the player’s subtotal by the parameter amt.

subtotal = subtotal + amt;

* public void updateScore() – increase the player’s score by the subtotal and then set subtotal to zero. Increment number of turns by one. For example:

score = score + subtotal;

* public void newGame() – reset the player’s score, subtotal and number of turns to zero.

**Test When Complete**

You can test your solution using Ch 16 zyProject Player.

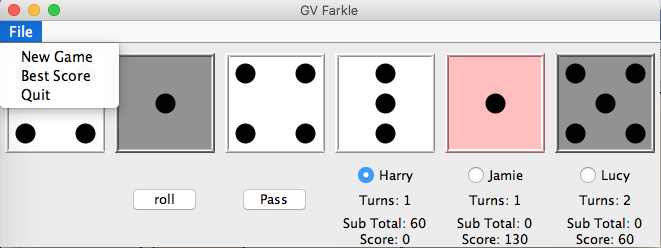
**Step 4: Use Existing FarkleStub**

YOU DO NOT WRITE ANY OF THESE. We are providing a simulated game for now so that you can focus on the GUI. You will eventually write your own game for Project 4. Right click on the FarkleStub.class file to download and Save As… on your computer in the folder for the newly created BlueJ project. Pay attention where it is saved and be sure to retain the correct file name. The icon might not appear on the BlueJ screen but your program will still be able to access it. If the following compiles within your GUI code then you are good to go:

private FarkleStub game;

* public FarkleStub() – creates the game object.
* public Player getActivePlayer() - returns current player.
* public boolean okToRoll() - returns true if it is legal for the player to roll the dice. Otherwise, returns false.
* public boolean okToPass() - returns true if it is legal for the player to pass the dice.
* public boolean checkFarkle() – for now, this method returns true 30% of the time regardless of the dice to support testing your GUI. This will be improved in Project 4.
* public void setActivePlayer(int id) – set the current player ID. Valid parameters include 1, 2 or 3 for a three-person game.
* publc boolean gameOver() - returns true if the game is over because the current player has earned enough points. Otherwise, returns false. For now, the winning score is 500 to make testing easier. This will be improved in Project 4.
* public ArrayList <GVdie> getDice() - returns an ArrayList of GVdie. You will display these dice in the GUI.
* public void resetGame() – reset game values.
* public void rollDice() - roll each of the dice that are not currently held. For now, scores are assigned somewhat randomly. This will be improved in Project 4.
* public void passDice() – player score is updated and dice are reset for next player. For now, scores are assigned somewhat randomly. This will be improved in Project 4.
* public Player getBestPlayer() – returns the player with the best score. For now, this always returns the same fictional character. This will be improved in Project 4.

**Sample GUI**  
There are no messages printed to the terminal window. All results appear within the GUI.



**Figure 1. This layout uses six columns for the dice. The right-most column has one die, a radio button and three labels (five rows). Pink dice are 'selected'. Gray dice have been scored.**

**Step 5: Create a GUI**

Create a new class called FarkleGUI. Adapt PizzaGUI from your recent lab activity by making the following changes. The goal is to make it look like Figure 1.

**Define instance variables**

* An object of type FarkleStub for the game back end and logic
* An object of type Player for the current player
* Menu items for New Game, Best Score and Quit (refer to Lab 11)
* Nine JLabels to display player information
* Three JRadioButtons for player names
* Two JButtons to roll and pass the dice

**setupMenu()**

A private helper method that creates the menu items (see GUI from lab).

* private void setupMenu()

**newGame()**

A private helper method that prepares for a new game.

* invoke the game’s resetGame()
* for each player, set the name, turns, subtotal and score. Adapt the following example:

theGame.setActivePlayer(3);

p = theGame.getActivePlayer();

player3Button.setText(p.getName());

score3Label.setText("Score: " + p.getScore());

subTotal3Label.setText("Sub Total: " + p.getSubtotal());

turns3Label.setText("Turns: " + p.getTurns());

**Within the constructor**

* Instantiate the game

theGame = new FarkleStub();

* Instantiate all buttons and labels
* Request the ArrayList of dice from the game object. The ArrayList can be local since it is not referenced anywhere else in the class.

ArrayList <GVdie> theDice = theGame.getDice();

* Place the dice in row 0 (spanning columns 0 - 5)
* Place the roll and pass buttons. Remember to register the action listeners.
* Create the button group with three JRadioButtons for each player (see GUI lab). Create and place the three radio buttons. Remember to register the action listeners.
* Place the nine labels for the three player scores and turns

**Within actionPerformed ( )**

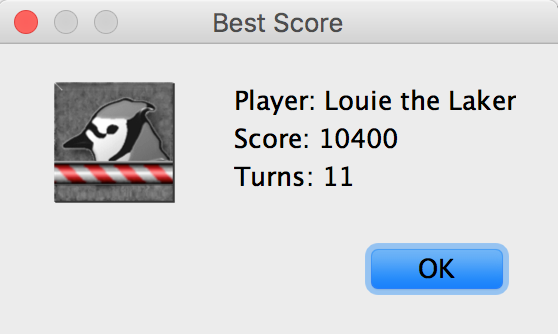
* Provide an if statement for the roll button. After rolling, ask the game if the player lost her turn and display an appropriate message.

if (e.getSource() == rollButton){

theGame.rollDice();

}

* Provide an if statement for the pass button and invoke the model’s passDice() method.
* Provide an if statement for the File / Quit menu option.
* Provide an if statement for the File / Best Score menu option. Ask the game object for the best player. Use a JOptionPane message dialog to display the player name, score and number of turns. For now, the best player is preset and will remain the same regardless of scores.



* Provide an if statement for the File / New Game menu option. Invoke your local helper method newGame().
* Provide if statements for each player radio button and set the current player as appropriate. The following example will be slightly different based on your variable names.

if (e.getSource() == player3Button){

theGame.setPlayer(3);

currentPlayer = theGame.getActivePlayer();

}

* Near the end of the method, if it is OK to roll then enable the Roll button. Otherwise, disable the button. Ask the game object if it is OK to roll.
* Near the end of the method, if it is OK to pass then enable the Pass button. Otherwise, disable the button. Ask the game object if it is OK to pass.
* Near the end of the method, provide an if statement for each radio button. Update the three player values: score, subtotal and turns. This is only done for the current player.

if (player2Button.isSelected()){

// update three information labels for the current player

score2.setText("Score: " + currentPlayer.getScore());

}

* Provide an if statement **at the end** of this method. Display a pop-up message if the game is over and disable both buttons. Include the current player’s name in the message. Use a JOptionPane.showMessageDialog.

if (theGame.gameOver()}{

// disable the roll and pass buttons

// display message

}



**Step 6: Testing a GUI**

It is difficult to automatically test a graphical user interface since it relies on someone clicking on buttons and selecting menu options in many unpredictable sequences. As an alternative, here is a checklist of actions your application should perform.

\_\_\_\_ Start game - does the game begin with blank dice, the roll button enabled and the pass button disabled? Do the radio buttons display player names?

**Check a normal turn**

\_\_\_\_ Click on roll – do the dice roll? Does the pass button become enabled?

\_\_\_\_ Select two dice – do they turn pink?

\_\_\_\_ Click on roll – do the pink dice turn gray and the remaining dice roll?

\_\_\_\_ Click on pass – do the nice turn blank and the player score is updated?

**Check for farkle**

\_\_\_\_ Change to second player – roll dice and select one die and continue to roll until player loses turn with a Farkle. This will happen somewhat randomly at this point. Does message display?

**Check for winner**

\_\_\_\_ Change to third player – continue playing and accumulating points until the player wins with at least 500 points. Does a message with the player’s name appear? Do both buttons disable?

**Check menus**

\_\_\_\_ Select Best Score from the menu – does information display for Louie the Laker with a best score of 10400 and 11 turns?

\_\_\_\_ Select New Game from the menu – do player scores return to zero? Do dice go blank?

\_\_\_\_ Select Quit from the menu – does the application quit?

**Grading Criteria**

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

**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 a screenshot of your GUI
* 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](http://www.cis.gvsu.edu/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.”
* Sample Output – provide a screenshot of the GUI on your cover page
* Source code – DO NOT print – upload to Blackboard your elegant source code for Player and FarkleGUI.
* Demo – be prepared to demo your working GUI during the lab.