CIS 162 Project 3
Farkle GUI
Section 04

Due Date

- at the start of lab on Monday 5, November

Before Starting the Project

- Play a few games of Farkle online and get a feel for the flow of the game. (One option is https://www.crazygames.com/game/farkle-master). For now, you do not need to know all the rules; you need only understand the basic flow of the game: Players roll until they either
    1. choose to pass and lock in the accumulated points, or
    2. make a bad roll called a "Farkle" and lose any points accumulated in that round.
- Read chapter 9 (GUI), chapter 10 (`ArrayList`) and 13 (arrays)
- Read this entire project description before beginning. (If you come into my office confused because you haven't read and followed all the directions --- including the one about reading the entire project description before beginning --- I will help you with a smile; but I will be silently writing a bad job reference in my head ☺)

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 Farkle. For Project 4, you will replace this simulated Farkle game with your own implantation.

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. (In other words, the GUI class should not know the rules, and should not maintain the score.)

GUI Features

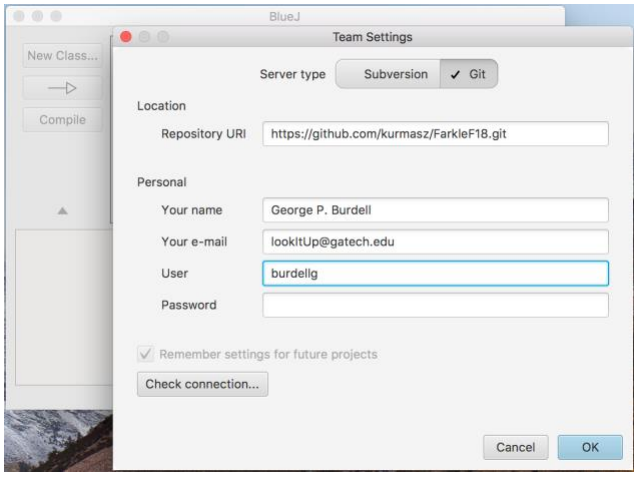
Your Farkle implementation will support multiple players using six dice.

- 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

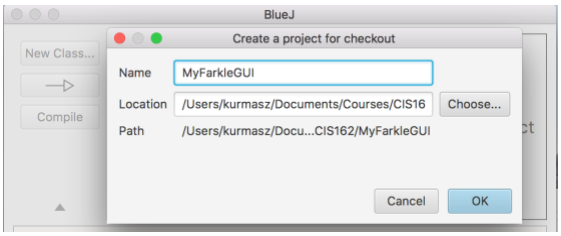
Step 1: Checkout the BlueJ project from GitHub

Instead of copying starter files by hand into a new BlueJ project, you can download the project and starter files from GitHub:

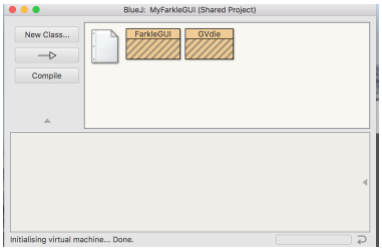
1. Launch BlueJ and go to Tools → Team → Checkout Project
2. Choose "Git" for the server type
3. Enter https://github.com/kurmasz/FarkleF18.git for the Repository URI
4. Enter anything you like for Your Name, Your e-mail, and User.



5. Hit "OK."
6. Select a name and location for the new project.



1. At this point, you should see your new project with two `.java` files. These two files should compile without error. The project directory also includes `FarkleStub.class`; but you won't see it in BlueJ.



Step 2: Use Existing GVdie

Rather than writing your own Die class, we are providing a completed class for you named `GVdie`. It should compile with no errors. Do not make any changes to this code; however, you are encouraged to read through the source code to see how it works.

Step 3: Create a Player class

Create a simple `Player` 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`). (This `Player` class is part of your model.)

Constructor

A *constructor* is a special method with the same name as the class and 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

FarkleStub is the other part of your model. It implements the rules of Farkle. ***YOU DO NOT WRITE ANY OF THESE METHODS.*** 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. 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.

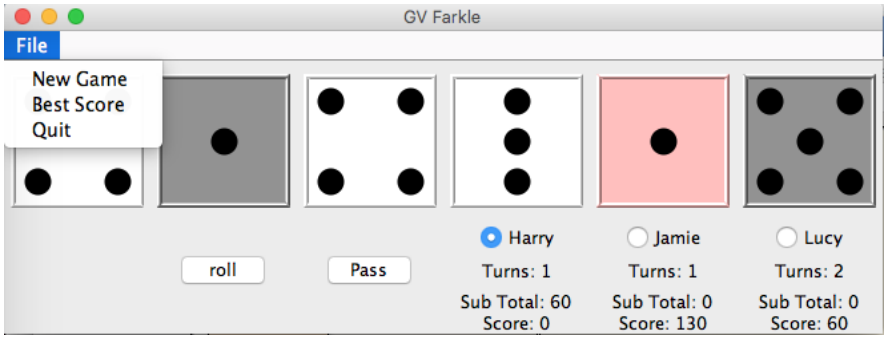


Figure 1. This layout uses six columns for the dice. The right-most column has one die, a radio button and three labels (five rows). Pink dice are 'selected'. Gray dice have been scored.

Step 5: Set up the GUI

Begin by setting up the GUI. Resist the temptation to implement some of the logic before the GUI is complete. Trust me: The next steps will be easier if you place all the components in your GUI first. If you like, you may adapt `PizzaGUI` from your recent lab activity. Your GUI should look like Figure 1.

You will want the following 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
- Nine `JLabels` to display player information (You may find it easier to create three arrays of three labels each: An array for the players' subtotals, an array for the players' number of turns, and an array for the player's score.)
- Three `JRadioButtons` for player names (Again, some of you may prefer to use an array.)
- Two `JButtons` to roll and pass the dice

Other suggestions

- Create a private helper method to set up the menu (as was done in the GUI lab).
- Instantiate the game in the constructor

```
theGame = new FarkleStub();
```

- Use a `GridBagLayout` like we did in the GUI lab.
- Request the `ArrayList` of dice from the game object. The `ArrayList` can be local to the constructor because 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.
- Create the button group with three `JRadioButtons` for each player (see GUI lab). Create and place the three radio buttons.

Step 6: Add functionality

IMPORTANT: This project will be *much* easier if you complete it feature-by-feature rather than method-by-method. (Trust me. I tried the method-by-method approach and quickly got confused.) Go through the list of features/behaviors below and get them working one-by-one. Yes, this means that you will occasionally backtrack and re-do some code; but, it will be much easier to follow the logic. Also, many software products in the real world are written using a similar feature-by-feature development approach.

To begin, you should have a complete GUI with labels, two buttons, three radio buttons, and three menu items that do nothing. One radio button should be selected.

1. Add an action listener to the Quit menu item that exists the game.

2. Disable the pass button and the radio buttons. Use Google to look up the API for the `JButton` class. Search through the list of methods for one that sounds like it will enable and/or disable the button. (Hint: Look in the `AbstractButton` section.)

   When this step is complete, only the roll button will be enabled. You won't be able to click on the pass button, or change the radio buttons.

3. Add an action listener to the roll button. For now, this listener should simply call the model's `rollDice()` method.

   When this step is complete, you should be able to repeatedly re-roll the dice. Remember: You must select one or more dice to "hold" before re-rolling.

4. Modify the action listener to check for a Farkle: If a roll results in a Farkle,
   a. display a message using a `JOptionPanel`, and
   b. disable the roll button.

   When this step is complete, users can only roll until they Farkle. When a user rolls a Farkle, they should (a) see a message, and (b) not be able to roll again. (Remember: The provided model class only simulates a game. It chooses at random whether a particular roll is a Farkle --- you won't be able to tell by looking at the dice.)

5. Modify the action listener to update the current player's display after each roll. Hint: Write a private helper method.

   When this step is complete, you should see the current player's updated subtotal after each roll. It should increase when there isn't a Farkle, and reset to 0 when there is.

6. Allow the user to pass the dice before rolling a Farkle.
   a. After rolling, ask the model whether the user is allowed to pass and enable/disable the pass button accordingly.
   b. Add an action listener for the pass button that
      i. Calls `passDice()` on the game object.

ii. Updates the current player's information (this is where the private helper method from the previous step is helpful)
iii. Enables the radio buttons so the next player can be chosen

c. Add an action listener to the radio buttons that does the following when the next player is chosen
   i. Enables the roll button
   ii. Disables the radio buttons
   iii. Tells the model who the new current player is.

When this step is complete, you should be able to pass if a roll doesn't result in a Farkle. After passing the dice, (1) The current player's subtotal should be added to her score. (2) The current player's subtotal should be re-set to 0. (2) The user should be able to select a new player. (3) The new player should be able to take his turn.

7. Change players when the user rolls a Farkle. Hint: The code to do this is already in place. You need only add one line to the block of code that runs when the user rolls a Farkle.

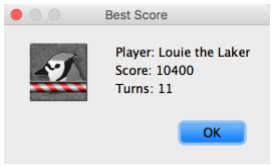
   When this step is complete, you will be able to continue the game after a user rolls a Farkle. When a user rolls a Farkle, his subtotal should be 0; and his score will not change.

8. Add a listener to the New Game menu item that resets the game. (Use the model's `resetGame()` method.)

   When this step is complete, selecting New Game from the menu should result in blank dice and all players having a score, subtotal, and turn count of 0.

9. Display a message when a player wins the game. Disable the roll and pass buttons.

10. Add a listener to the Best Score menu option. Ask the game object for the best player and 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.



Step 7: 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.
- 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 - a printout of your elegant source code for `Player` and `FarkleGUI`.
- Demo – be prepared to demo your working GUI and make minor changes at instructor's request