



Advanced Java ICP 2151

Simple Dice Game with GUI Laboratory 4

Steve Marriott

Assessed Laboratory Work

In this session we put together some of the ideas covered in the previous weeks and develop a Java application which plays the popular American dice game known as “Craps”. The version of the game we shall develop is played against the computer by a single player interacting by means of a graphical user interface.

The Game of Craps

Craps is defined by a few straightforward rules described below. Before the game starts the player (i.e.) you is said to have a *point* value which is initially set to zero. The game now starts with a pair of unbiased dice being rolled. The sum of the dice faces is then calculated. Play then proceeds as follows:

- If the sum is 7 or 11 the player wins;
- If the sum is 2, 3 or 12 then the player loses and the computer wins;
- If the sum is 4, 5, 6, 8, 9 or 10 the *point* is set to the *sum*.

After the first throw if neither party has won then the dice are thrown again repeatedly until either:

- The sum of the dice is *equal* to your point in which case the player wins;
- The sum of the dice is *equal* to 7 in which case you lose and the computer wins.

The game ends as soon as the player or the computer wins.

Remember that a primary design goal for a graphical user interface is “ease of use”, so try to achieve that goal in this assessment. The operation of the game is shown below:

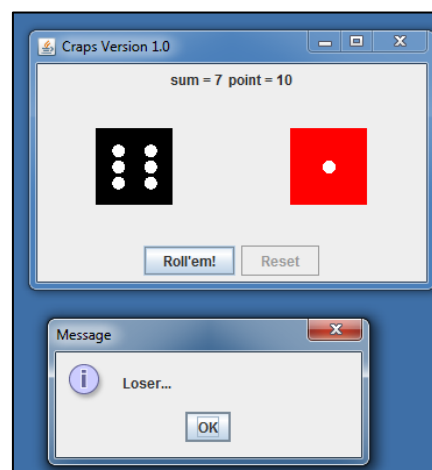


Figure 1 Playing Craps and Losing

The player has just lost because the sum of the dice faces is equal to seven and this signalled by the use of a pop-up message.

Part 1: Placing the Dice (20%)

The first thing to do is to create a fixed size frame capable of displaying two dice faces. Your program should be able to generate something like the following:

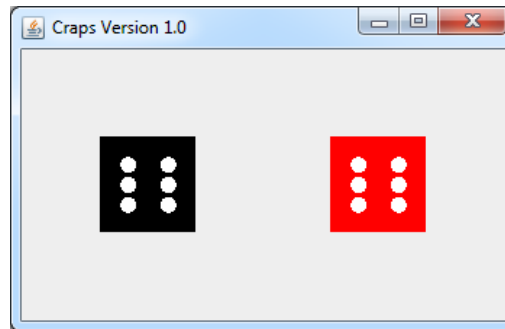


Figure 2 Placing Two Components in A Frame

At this stage you are not required to provide any facility for changing the value of the dice face.

Part2: Adding a Throw Button (20%)

Having placed the dice face images inside a frame provide a throw button which when clicked will cause the two dice to be thrown and the dice face components repainted.

Your program should generate something like the following:

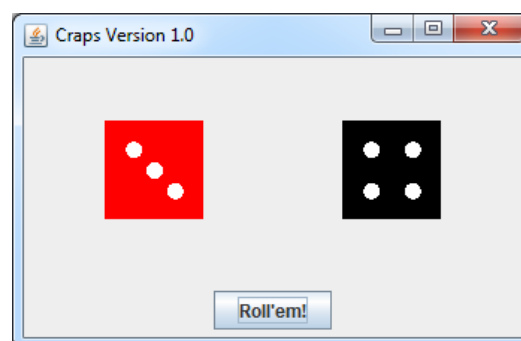
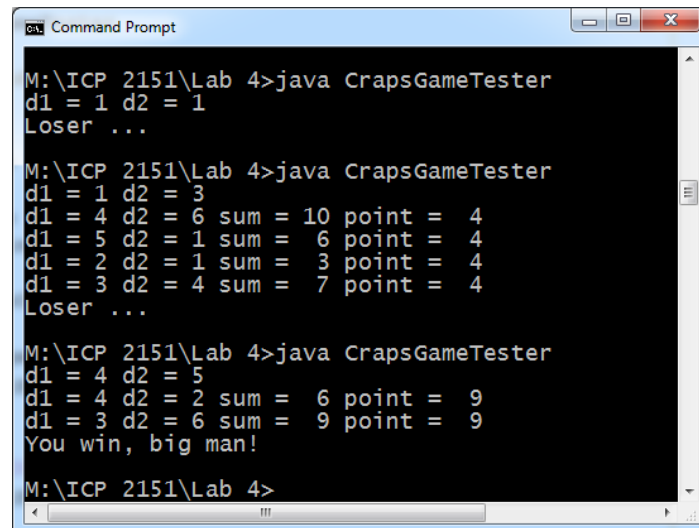


Figure 3 Rolling the Dice

Note that the program at this stage must enable the user to *repeatedly* throw the dice and see the new values.

Part 3: Define the Game Logic(20%)

Develop a class called **CrapsGame** and a test driver called **CrapsGameTester**. These two classes should support a *console* version of the game of Craps. Output from the test program is shown below:



```
Command Prompt
M:\ICP 2151\Lab 4>java CrapsGameTester
d1 = 1 d2 = 1
Loser ...

M:\ICP 2151\Lab 4>java CrapsGameTester
d1 = 1 d2 = 3
d1 = 4 d2 = 6 sum = 10 point = 4
d1 = 5 d2 = 1 sum = 6 point = 4
d1 = 2 d2 = 1 sum = 3 point = 4
d1 = 3 d2 = 4 sum = 7 point = 4
Loser ...

M:\ICP 2151\Lab 4>java CrapsGameTester
d1 = 4 d2 = 5
d1 = 4 d2 = 2 sum = 6 point = 9
d1 = 3 d2 = 6 sum = 9 point = 9
You win, big man!

M:\ICP 2151\Lab 4>
```

Figure 4 Console Version of Craps

In the display above, the identifiers **d1** and **d2** refer to the two dice. To help you with this part of the assessment the source code for the test driver is listed below. It is recommended that you use this code to identify and name the methods required for the implementation of the **CrapsGame** class.

```
public class CrapsGameTester
{
    public static void main(String args[])
    {

        CrapsGame game = new CrapsGame();

        String message = game.doFirstRoll();
        System.out.println(message);

        while(!game.isOver())
        {
            message = game.rollAgain();
            System.out.println(message);
        }

        if(game.isWon())
            System.out.println("You win, big man!");
        else
            System.out.println("Loser ...");
    }
}
```

The string **message** contains a *formatted* string recording part of the game state, namely the value of the dice, the sum of the dice values and the point.

It is important to understand that because of the design of CrapsGame the class can be used to support a console application, as well as GUI based version of the game.

Part 4: Attaching the Game to the Frame(20%)

Having tested the game logic it is time to *connect* the game to the GUI. This is relatively simple because it mostly involves modifying the **actionPerformed()** method associated with the listener for **Roll'em** button.

When testing the attachment it is a good idea to use console output to see what is going on.

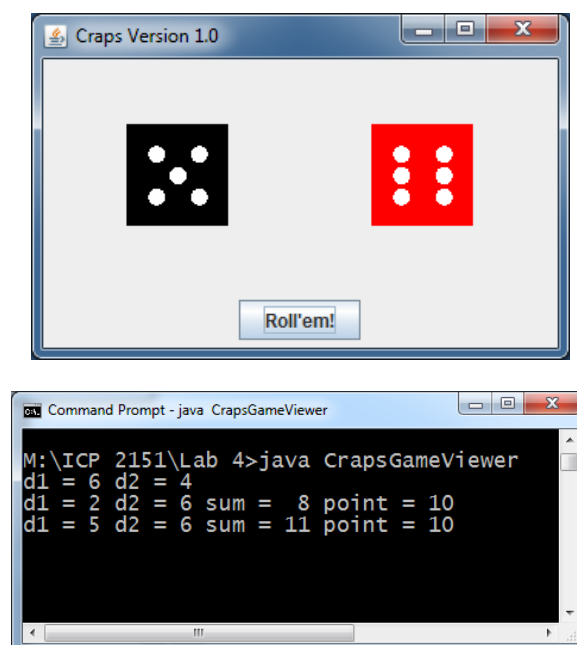


Figure 5 Using the Console to View Game State

When the game is over – because the player has either won or lost - the program should automatically detect the situation and close the application with a call to **System.exit()**.

Part 5: Displaying the Game State (10%)

This is the final part of the assessment and involves adding some extra features to the interface. The first extra feature to be provided is a pair of labels to help the player keep track of the game. The labels are used to display the sum of the dice and the value of the players point. This is illustrated below where the initial values are displayed:

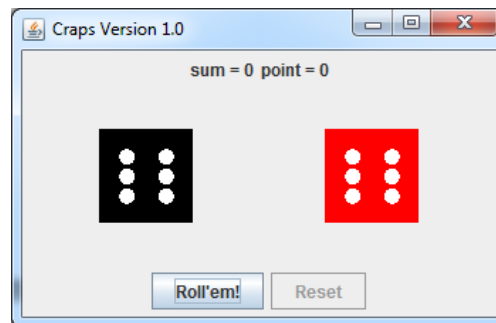


Figure 6 Initial value of sum and point

The player now rolls the dice and the sum and point are calculated and the new values displayed:

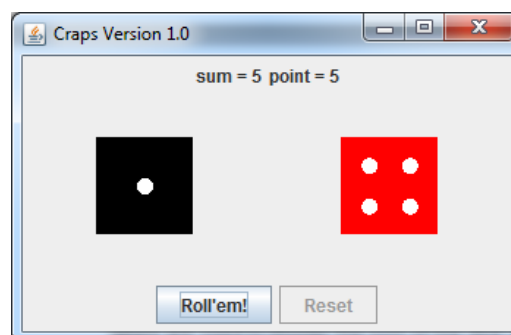


Figure 7 Updating sum and point

The player now rolls again but this time only the sum value is updated.

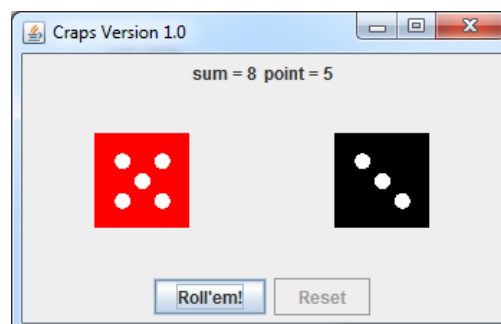


Figure 8 Updating sum

The process of updating the **sum** value should now continue until the end of the game. When the game finishes the reset button should be enabled and when clicked will reset the values of **sum** and **point** and set both dice face to display the value six.

Part 6: Announcing the Result(10%)

To complete this simple application use **JOptionPane** objects to display a win or a lose message. This idea is illustrated below:

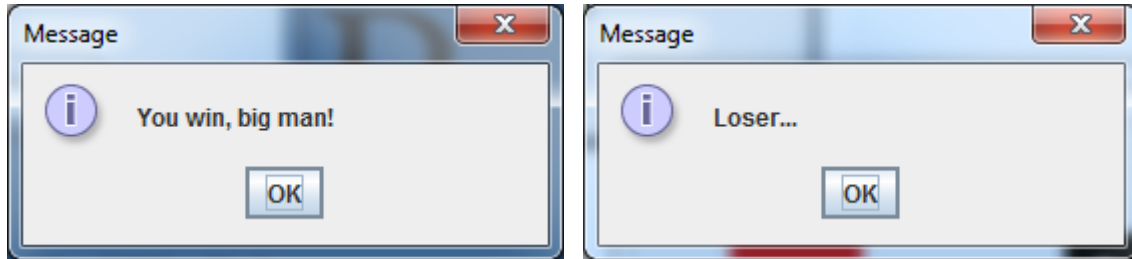


Figure 9 Win and Lose Message Dialogs

At the end of the game one or other of the two message dialogs above should appear.

Submission

Use Blackboard to submit your source code files. These files must

- Contain a program header (with your name)
- An appropriate level of Javadoc style comments
- Follow a consistent style of indentation
- Follow the usual Java conventions for class and variable names

The deadline for submission is posted on Blackboard. Late submissions will be penalised in line with School policy.

Marks for this laboratory exercise are awarded for the following:

- Two dice images rendered
- Dice images controlled by **JButton**
- Controls placed in frame
- Game plays correctly
- Overall “look and feel”
- Program structure
- Comments and layout
- Conceptual understanding

When submitting work it is your responsibility to ensure that all work submitted is

- The work requested
- Entirely your own work
- On time

Please note that there are severe penalties for submitting work which is not your own. If you have used code which you have found on the Internet then you *must* signal that fact with appropriate program comments.

Note also that to obtain a mark you *must* attend a lab session and be prepared to demonstrate your program and answer questions about the coding. Non-attendance at labs will result in your work not being marked.

Steve Marriott