# CS 114 Lab 19

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Task 1—Rock-Paper-Scissors (one hand)

Rock-Paper-Scissors (じゃんけん) is a two-player game normally played with your hands. You and an opponent (on a count of 3) each make your hand in the shape of a rock (fist), scissors (two fingers), or paper (palm flat). If they’re the same, it’s a draw, and you play again. Otherwise, the winner is determined by a simple rule. We will build a game where two computer opponents play against each other. (Maybe some other day we’ll make a human player against the computer.)

1. Download “Lab 19.zip” from Blackboard. Open the package.bluej file in BlueJ.
2. The Hand class represents a player’s hand and the formation it is currently in. Open the file and take a look at it. It has one instance variable, formation. It is an int and takes on one of three values: 0, 1, and 2. These values are stored in constants so we can refer to them by name.
3. The Hand class also has a constructor. What is the state of the hand formation after a Hand object has been instantiated?
4. The next things in the Hand class are an accessor and a mutator method for the formation variable. They should be self-explanatory, but the mutator (setFormation) is missing one thing. The comment says the method is supposed to throw an exception if the parameter value is not a legal play (i.e. 0, 1, or 2), but it sets the value no matter what. Fix this by adding an appropriate if statement and an appropriate throws statement.
5. Two more methods in the Hand class that are already complete are pickRandom and toString. Familiarize yourself with these. Why does pickRandom work? (Why does it always set the hand to rock, paper, or scissors?)
6. Finally, the beats method needs to be implemented. It will be the primary means of determining who wins in a Rock-Paper-Scissors game using this class. Implement this method. (Note that this method does not print anything—it just returns true or false.)
7. Now test the class in BlueJ. Create an instance of the Hand class by right-clicking in BlueJ and choosing “new Hand.” Then try its various methods. Methods that require a parameter will ask you to supply one in a little dialog box. Methods that return a value will pop up the result in a dialog box. Look at the value to see if it makes sense. In particular, be sure to test the beats method with several possible input values. It should return true or false correctly.
8. When it works, show the code and the result to the instructor: \_\_\_\_\_\_

## Task 2—Rock-Paper-Scissors (the game)

1. The RockPaperScissorsGame class is a panel for letting a user watch the computer play the game against itself. It has a single button that the user clicks to start each round. The computer players reveal their moves, and the score is tallied. It makes extensive use of the Hand class.
2. Open the class and take a look at the code. Most of this is set up for you. Run the main method of the Viewer class. Compare how it looks and how it works to what you see in the code.
	1. Clicking the button doesn’t seem to do anything. Why?
	2. The labels both say “rock.” Why is that?
	3. Explain why the “Go!” button is above the labels.
3. The missing piece to this game is the runOneRound() method. It has comments indicating what needs to be done.
	1. How would you “Generate random moves for each player”?
	2. How will you “Check to see who won”?
	3. Why does it say “(if any)” after “see who won”?
	4. What are the three parts of the labels? Which parts have to be updated?
4. Implement the runOneRound() method according to the comments in the code and your ideas above.
5. Run the Viewer program to see if it plays the game correctly. Show the code and the result to the instructor: \_\_\_\_\_\_

### Turn in this paper at the end of lab.

## Lab 19 Homework—Risk

For this project, we’ll simulate one battle from the Parker Brothers strategy game Risk. In the original game there are several territories, each under the control of a general who is trying to conquer the entire world. We’ll simplify the game for now by considering only two territories. Battle results are determined by dice rolls. The rules are a little complex, so we will only consider the case where the attacker has two dice and the defender has one die. Both players roll at the same time. The attacker’s roll value is the max (not the sum) of his two dice, and the defender’s roll value is simply the value on her one die. However, ties go to the defender, so it balances out a little. Each general has a certain number of armies to start with in his or her territory, and whoever loses the roll loses one army. The battle continues until the attacker decides to quit (or is forced to quit by having fewer than 3 armies) or the defender’s armies are completely wiped out.

1. Open the Die class in the Lab 19 BlueJ project. Familiarize yourself with this class.
2. Missouri is attacking Kansas! Everyone, to arms! The RiskGame panel is very similar to the RockPaperScissorsGame panel from the lab. (Look there for example code.) Your task is to implement everything necessary in this class for the game to work. Some of it is started for you.
3. I expect the game to play like this: I run the main method of the Viewer class. I see Kansas and Missouri and the number of armies currently in each. There is a button. I click the button to see the results of the three dice. The number of armies in each state is also updated correctly. (Since ties go to the defender, one or the other state will always lose one army each round.)
4. Declare the instance variables you will need under the “Game objects and state variables” comment. You will need
	1. Two dice for player 1 (Missouri)
	2. One die for player 2 (Kansas)
	3. An int for each player that holds the number of armies.
5. Initialize the above variables under the “Set up player 1” and “Set up player 2” comments:
	1. Construct 3 dice and assign them to the variables.
	2. Set the initial number of armies for each state. Determine these values randomly, between 5 and 25.
6. Add a “battle” button:
	1. Declare an instance variable under the “GUI elements” comment.
	2. Create a private class that extends ActionListener. The actionPerformed method should call the runOneRound method (just like the code from the lab).
	3. Under the “Set up battle button” comment:
		1. Construct a new JButton and assign it to the battle button instance variable.
		2. Add the listener to the button.
		3. Add the button to the panel.

## Lab 19 Homework—Risk, continued

1. Implement the runOneRound() method, following the comments:
	1. Roll the dice.
	2. Check to see who won and decrement armies accordingly.
	3. Update the labels. Each label should show the name of the state, the current number of armies, and the die roll values (one for Kansas, two for Missouri).
2. When the battle is over, do the following:
	1. Disable the battle button (see step 6d of the Voting Machine of Lab 18).
	2. Pop up a message dialog box (use JOptionPane.showMessageDialog) saying who won.

The battle is over either when Kansas is reduced to 0 (defeated, Missouri moves in) or Missouri is reduced to 2 (no longer powerful enough to invade).

1. Upload the RiskGame.java file to Blackboard.
2. This is individual work. You can talk to each other about ideas, but you must write your own code.
3. Grading will be based on correctness (how well it fulfills the requirements given) and style (code formatting—tabs, variable names, etc.).

### Grading

Correctness

initial number of armies /5

button appearance /5

dice rolls /10

army updates /10

label appearance /10

battle over /10

Coding style

header comments /10

code comments /5

indentation /5

variable names /5

other (see notes) /5

Total /80

### Due at 11:59 p.m. Mon., 8 Dec. 2014. Don’t delay! The end of the semester is upon us.