Risk Dice

Professor Don Colton

Brigham Young University—Hawaii Campus

1 Overview

Risk (TM) is a board game sold by Parker Brothers, a division of Hasbro. Wikipedia has a good article describing the game.

`http://en.wikipedia.org/wiki/Risk_(game)`

A key part of this game is the battle of army against army. The winner is determined by a series of dice rolls. Our program will roll the dice and determine the results of the battle.

1.1 Input

The attacking army has some number of military units. The defending army has some number of military units.

1.2 Battle

A battle consists of one or more rolls of the dice.

All but one of the attacker's units can participate in the battle. That last unit must stay behind after the battle to defend the country it is in. The attacker can roll 1, 2, or 3 dice, but no more than the number of units available to participate. For simplicity in our program the attacker will always roll the maximum number of dice possible. In the real game, the attacker can choose to roll a smaller number, but must roll at least one die.

All of the defenders units can participate in the battle. The defender can roll 1 or 2 dice, but no more than the number of units they have. For simplicity in our program the defender will always roll the maximum number of dice possible. In the real game, the defender can choose to roll a smaller number, but must roll at least one die.

For simplicity in our program the battle will continue round after round until the attacker has no more units with which to attack, or until the defender has no more units with which to defend. In the real game, the attacker can quit the battle earlier.

1.3 Output Loop

Each roll of the dice will be introduced by a statement of how many armies each side has, and how many are participating in the battle.

For example, if the attacker has 10 units and the defender has 5 units, we would print the following statements.

```
Attacker has 10.  Defender has 5.
```

```
Attacking with 3.  Defending with 2.
```

Next we roll the dice and sort and announce the results.

```
Attacker rolls 6 5 2
```

```
Defender rolls 5 5
```

Next we announce the effects on the armies.

```
6 against 5: defender loses 1.
```

```
5 against 5: attacker loses 1.
```

We repeat these steps until attacker is down to 1 or defender is down to zero.

1.4 Output Conclusion

Finally, we report the new composition of the armies.

```
Attacker has 3.  Defender has 0.
```

```
Attacker wins.
```

1.5 Statistical Summary

We will also report the number of times each side rolled each number. This will help us see whether the dice are fair or biased.

```
Statistical Summary

Attacker rolled 6 13 times.

Attacker rolled 5 13 times.

Attacker rolled 4 13 times.

Attacker rolled 3 13 times.

Attacker rolled 2 13 times.

Attacker rolled 1 13 times.

Defender rolled 6 13 times.

Defender rolled 5 13 times.

Defender rolled 4 13 times.

Defender rolled 3 13 times.

Defender rolled 2 13 times.

Defender rolled 1 13 times.
```

2 Tasks

This entire project is expected to take about two weeks (six hours) of class time, plus some amount of time working outside of class.

To make things more managable, we have broken the project up into smaller tasks that gradually build toward the finished project.

2.1 Roll

Create a subroutine named `roll` that returns a random number between 1 and 6. This should simulate a fair die, which means the numbers are uniformly distributed (each number is equally likely).

Test it by calling it 500 times and printing the results of each roll.

2.2 Statistics

Create a subroutine named `statistics` that takes as input a list of dice rolls and returns the count for how many times each number was rolled.

Test it by calling roll 500 times and feeding those results into statistics, and then printing the results.

2.3 Attacking

Create a subroutine named `attacking` that accepts as input the number of units the attacker has and returns the number of units that will participate in the next round of the battle. The formula is A-1, where A is the number of units the attacker has. After calculating A-1, if the number is more than 3, just return 3.

Test it by calling attacking with all numbers from 1 to 10 and printing the results.

2.4 Defending

Create a subroutine named `defending` that accepts as input the number of units the defender has and returns the number of units that will participate in the next round of the battle. If the defender has only one unit, the answer is 1. Otherwise the answer is 2.

Test it by calling defending with all numbers from 1 to 10 and printing the results.

2.5 Skirmish

Create a subroutine named `skirmish` that accepts as input the dice rolls from the attacker and the dice rolls from the defender. It sorts each list and prints out who won and lost. It returns two numbers: the number of units lost by the attacker and the number of units lost by the defender.

2.6 Battle

Create a subroutine named `battle` that accepts as input the starting number of attacking units and defending units. It calls `attacking` and `defending` to determine how many dice to roll. It calls `roll` the appropriate number of times and saves the results.

It calls `skirmish` with those results and receives the units lost. It updates the total units still in play. Then it repeats until there is a winner.

3 GradeBot

To prove that your program is working exactly as needed, submit it to GradeBot. GradeBot will replace your `rolls` program with its own so it can control all the outcomes.