**CS 10 - Assignment 5: Pig - Monte Carlo Method**

**Collaboration Policy**

We encourage collaboration on various activities such as lab, codelab, and textbook exercises. However, **no collaboration between students is allowed on the programming assignments**. Please be sure to read and understand our full policy at: [Full Collaboration Policy](https://docs.google.com/document/d/1WyzL3qvKLrC1UCRf178b_wYWQmEZlhDObFNFb79U63I/edit?usp=sharing)

**Submission Instructions**

Submit to [R’Sub](https://galah.cs.ucr.edu) testing, feedback and grading.

**Assignment Specifications**

We are going to use the [Monte Carlo Method](http://en.wikipedia.org/wiki/Monte_Carlo_method) to determine the probability of scoring outcomes of a **single turn** in a game called Pig.

**Your Task**

For this assignment you will simulate a given number of **hold-at-N turns** of a game called Pig, and report the estimated probabilities of the possible scoring outcomes. You are **NOT** implementing the game of Pig, only a *single turn* of this game and simulating this single turn over and over. The value of **N** will be acquired via user input, as will the number of repetitions.

**You must implement two functions to help implement the program. These two functions are outlined below.**

**What is Pig?**

[Pig](http://cs.gettysburg.edu/projects/pig/piggame.html) is a folk dice game every single turn boils down to two simple rules where a play rolls or does not roll. Each time a roll of the die occurs, an action is taken based on the die value. For our simulations, we will implement a specific strategy such that once our minimum turn score is reached the turn will end (opt to hold).

In summary, a turn looks like:

* **roll (or roll again)** - a roll of the die occurs
* **2 - 6**: the number is added to the current turn score; the turn continues
* **1**: the player loses all points accumulated in the turn (i.e. scores a 0); turn ends
* **hold** - The turn ends. A hold may occur for multiple reasons.

**Before moving on, ask yourself:** what are the **ways** to end a turn**?**

**Hold-at-N Turn Strategy**

A good strategy to help decide when to hold and when to roll is the “**hold-at-N strategy”**:   
the player chooses a number, N, that will hopefully both maximize their turn score while minimizing their chances of losing that score by rolling a 1; as soon as their current turn score reaches (or passes) N, the player holds.

We are going to test this strategy for different values of N, which will be supplied by user input, by simulating a number of turns (which will also be supplied by user input). Obviously, the larger the number of simulations, the better the estimate of probabilities.

For instance, suppose the user asks the program to test the strategy for N = 21.  
We throw the die for a turn (“simulate a turn”), and get the following rolls:

Roll 1: 5 - current turn score = 5  
Roll 2: 4 - current turn score = 9  
Roll 3: 4 - current turn score = 13  
Roll 4: 3 - current turn score = 16  
Roll 5: 3 - current turn score = 19  
Roll 6: 4 - current turn score = 23

Now, we end the turn because our score is at least the hold value; we have a turn score of 23.

**Functions**

The first function will generate a random integer, given a specified minimum and maximum integer for the range. The second function will execute a single turn of Pig and return the score for that specific turn. R'Sub will test these functions individually, you should implement your functions first and then move on to the core main program that utilizes your functions; testing with R'Sub to make sure your functions are correct prior to moving on.

**genRandInt**

* **name:** genRandInt
* **return type:** integer
* **parameters:** two integers (order matters)
  + first: the minimum possible random number for the requested range
  + second: the maximum possible random number for the requested range

**singleTurn**

* **name:** singleTurn
* **return type:** integer
* **parameters:** one integer representing the hold value for a single turn

**Random Seed Requirement**

We want you to submit to R'Sub with srand(time(0)); as the seed. However, when you are testing your program, you will want to compare your program’s output with the sample runs, so you will want to work with the *exact same* random values that are used in our solution. To do this you have to seed rand with 333: srand(333);

A note to those not working within C9. The random numbers generator by different systems (Windows, Linux, OS X) may be different as the implementation for generating pseudo random numbers is not exactly the same, so if you are developing outside C9 and are not getting precisely the displayed example, try either running on C9 or submitting to R'Sub.

**When debugging your program using srand(333), the first 25 rolls of the die should be:**

5 4 4 3 3 4 5 2 3 6 5 2 4 1 5 5 3 3 6 4 4 3 4 2 6

**Input Requirements**

* Enter a single positive integer indicating the number at which to hold.
* Enter a single positive integer indicating the number of turns to be simulated.
  + **Larger numbers will tend to yield better estimations but take longer to execute.**
  + **We test with both small and large numbers, so testing may take some time.**

**Output Requirements**

* + Prompt the user with: "What value should we hold at? "
  + Prompt for number of simulations: "Hold-at-N turn simulations? "
  + Output a blank line between the input prompt and table output.
  + On the next line, print "Score" and "Estimated Probability" separated by a tab (‘\t’).
  + After the simulations, print a table line for each score outcome that occurred, in increasing order of score.
    - **For each score** outcome, **print the score, a tab**, and the **fraction of turn simulations** that yielded that score **rounded to six digits after the decimal place**.

**Common error**

Make sure you count the results from *every* simulation - it is easy to accidentally omit the first or last run from your calculation. This will be barely noticeable if you have 1,000,000 simulation runs - but it makes a big difference if you only have 3 or 4!

**Example Runs** (User input has been **bolded and underlined** to help differentiate typed input from program output.)



|  |  |
| --- | --- |
| user@cs10: $ g++ pig.cpp  user@cs10: $ run a.out  What value should we hold at? **21**  Hold-at-N turn simulations? **1**  Score Estimated Probability  0 0.000000  21 0.000000  22 0.000000  23 1.000000  24 0.000000  25 0.000000  26 0.000000  user@cs10: $ | user@cs10: $ g++ pig.cpp  user@cs10: $ run a.out  What value should we hold at? **21**  Hold-at-N turn simulations? **2**  Score Estimated Probability  0 0.000000  21 0.500000  22 0.000000  23 0.500000  24 0.000000  25 0.000000  26 0.000000  user@cs10: $ |
| user@cs10: $ g++ pig.cpp  user@cs10: $ run a.out  What value should we hold at? **21**  Hold-at-N turn simulations? **3**  Score Estimated Probability  0 0.333333  21 0.333333  22 0.000000  23 0.333333  24 0.000000  25 0.000000  26 0.000000  user@cs10: $ | user@cs10: $ g++ pig.cpp  user@cs10: $ run a.out  What value should we hold at? **21**  Hold-at-N turn simulations? **4**  Score Estimated Probability  0 0.250000  21 0.250000  22 0.250000  23 0.250000  24 0.000000  25 0.000000  26 0.000000  user@cs10: $ |
| user@cs10: $ g++ pig.cpp  user@cs10: $ run a.out  What value should we hold at? **17**  Hold-at-N turn simulations? **10000000**  Score Estimated Probability  0 0.570301  17 0.114186  18 0.108193  19 0.083909  20 0.062619  21 0.040803  22 0.019988  user@cs10: $ |  |

Notice that the results all have **six digits to the right of the decimal** *(what formatting “manipulators” did we use to achieve that?)*

Also note that in all cases (as we would expect!) the total probability - i.e. the sum of the probabilities of all possible scores - will be 1.000000 (100%)

So, for example, the 1 run simulation example estimates a 100% probability for the score that was actually rolled in the run.