**CS 10 - Assignment 8: Monte Carlo - Board Game**

**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**

Once again, we are going to use the [Monte Carlo Method](http://en.wikipedia.org/wiki/Monte_Carlo_method), this time to determine the different frequencies of landing on each spot of a board in a single circumnavigation of the game board.

**The Board**

You should now be learning to make your programs as general as possible, so for this assignment, you have to be able to make your calculations work for any board size and configuration: all you know about it is that all the "cells" (the squares on which a player can land) are evenly distributed around the edge of an n-sided polygon, k cells per side. For our purposes, we shall number the cells 1 through (n \* k), the last cell is the original starting position.

**The Simulation**

A player rolls a pair of 6-sided dice, and moves that many cells from the start spot; this process is repeated as many times as required until the roll places the "piece" past the starting position.

Since we are modelling only a single player, there are no turns: we simulate a single player going around the entire board once. The starting space is the last possible spot a player can land in a single rotation.

**Your Task**

You will model the board as a vector of size (n \* k), with start spot corresponding to the last cell of the board vector. *Note that although you will be working with indices 0 through (n \* k - 1), you will report the cells as numbers 1 through (n \* k), as this is how a player would think about the board.*

As your player traverses the board, you will record each cell he/she lands on. You will perform multiple simulations a single board rotation, accumulating in each vector cell the number of times the player lands on the corresponding board space.

**You will implement several functions for this program. As you have learned, you must understand how to test each function with a "test harness". We do not submit the harnesses to R'Sub, R'Sub already has several to test with.**

**Functions**

We list two functions that are required, we provide an idea for one optional function and you can implement other additional functions that you find useful. You should write function comments for each of the functions you define, containing an @brief, @param and @return similar to those we have previously provided for you. R'Sub will test these functions individually - make sure they are correct before moving on.

**The first function** will simulate the dice roll. Once again we want the function to be as general as possible, so we will parameterize both the number of dice and the number of sides on a dice.

**The second function** will return the cell with most landings within a closed interval of board spots.

**rollNDice**

* **simulates rolling N dice, each of which has the specified number of sides**
* **parameters:** two integers by value:
  + first integer: the number of dice to roll, by value
  + second integer - the number of sides on the dice, by value
* **return type:** integer, the summation of all N dice rolls

**mostLandings**

* **returns the cell on the board with the most "landings", within an inclusive interval**
* **parameters:** a vector<int>; two integers
  + first parameter: the game board vector, by const reference
  + second parameter: the start of the interval to be tested, by value
  + third parameter: the end of the interval to be tested, by value
* **return type:** integer, the first cell in the interval with the most landings

Optionally, you may want (in addition to other functions you think of):

**printBoard**

* **procedure, prints out game board, showing number of landings on each cell**
* **parameters:** a vector<int>
  + board - the game board vector, by const reference
* **return type:** void

**Random Seed Requirement**

We want you to submit to R'Sub with srand(time(0)); as the seed. However, to reproduce the results displayed in the examples you you have to seed rand with 333: srand(333);

A note to those not working within C9. The random numbers generator on different systems (Windows, Linux, OS X) may produce different results even when using the specified seed value.

**When debugging your program using srand(333), the first 24 rolls of a die on c9 (linux) are:**

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

**Input Requirements**

* Prompt the user for the board configuration: number of sides, number of cells per side
* Prompt the user for the number of simulations to run.

**Output Requirements**

Use the mostLandings function to help you find and report the cell that has the most landings on each side of the board. As shown in the examples, each side of the board is reported. You will need to write a loop to print out this information and the loop should use simple mathematics to calculate the beginning and ending spots based on the side number.

**Explicit Example**

For our explicit example we will utilize the board game Monopoly. The board for Monopoly has 4 sides of 10 spaces, so our vector needs 40 cells. The start space in Monopoly is "Go", but "Go" will be the last spot represented by the last cell in the vector as the first board space is the spot immediately after "Go". We are not incorporating the values of the board in the simulation (so landing on jail does not send you to jail).

Based on the sides and the spots per side we can determine that side 1 is spots 1-10, side 2 is spots 11-20, side 3 is spots 21-30, and side 4 is spots 31-40. In the game Monopoly, a player rolls two six sided dice. We will use the 25 rolls of srand(333) within our example simulations.

We show the dice rolls (based on the 24 values), the accumulation of the dice rolls and the vector cell that would be incremented based on the dice rolls. We show two simulations, so if you state 1 simulation then only simulation 1 would run but if you state 2 simulations both simulation 1 and simulation 2 would run before the program completes. The first two example runs correspond to our specific Monopoly example.

**Simulation 1**

|  |  |  |
| --- | --- | --- |
| Dice Rolls:   1. 5+4 -> 9 2. 4+3 -> 7 3. 3+4 -> 7 4. 5+2 -> 7 5. 3+6 -> 9 6. 5+2 -> 7 | Accumulation after roll:   1. 9 2. 16 3. 23 4. 30 5. 39 6. 46 | Vector cell increased:   1. 8 2. 15 3. 22 4. 29 5. 38 6. Beyond "Go" (40) |

**Simulation 2**

|  |  |  |
| --- | --- | --- |
| Dice Rolls:   1. 4+1 -> 5 2. 5+5 -> 10 3. 3+3 -> 6 4. 6+4 -> 10 5. 4+3 -> 7 6. 4+2 -> 6 | Accumulation after roll:   1. 5 2. 15 3. 21 4. 31 5. 38 6. 44 | Vector cell increased:   1. 4 2. 14 3. 20 4. 30 5. 37 6. Beyond "Go" (40) |

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

|  |
| --- |
| user@cs10: $ g++ boardGame.cpp  user@cs10: $ run a.out  How many sides of the board are there? 4  How many spots are on each side? 10  How many simulations? 1  The following spots on each side have the most landings:  On side 1, spot 9 has the most landings: 1  On side 2, spot 16 has the most landings: 1  On side 3, spot 23 has the most landings: 1  On side 4, spot 39 has the most landings: 1 |
| user@cs10: $ g++ boardGame.cpp  user@cs10: $ run a.out  How many sides of the board are there? 4  How many spots are on each side? 10  How many simulations? 2  The following spots on each side have the most landings:  On side 1, spot 5 has the most landings: 1  On side 2, spot 15 has the most landings: 1  On side 3, spot 21 has the most landings: 1  On side 4, spot 31 has the most landings: 1 |
| user@cs10: $ g++ boardGame.cpp  user@cs10: $ run a.out  How many sides of the board are there? 1  How many spots are on each side? 11  How many simulations? 3  The following spots on each side have the most landings:  On side 1, spot 9 has the most landings: 2 |
| user@cs10: $ g++ boardGame.cpp  user@cs10: $ run a.out  How many sides of the board are there? 6  How many spots are on each side? 12  How many simulations? 10000000  The following spots on each side have the most landings:  On side 1, spot 7 has the most landings: 1821959  On side 2, spot 16 has the most landings: 1482794  On side 3, spot 26 has the most landings: 1437018  On side 4, spot 44 has the most landings: 1430447  On side 5, spot 49 has the most landings: 1430818  On side 6, spot 65 has the most landings: 1430729 |