Computer Laboratory 2

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Introduction

In this lab we are going to write a computer program to simulate a simle dice game: “LCR”. The LCR game is a simple dice game, the formal rules will be explained later in this document, but the aspects that make it interesting for us that *players do not make any choices.* Because of this, we can fully simulate this game without any need for a true AI to take the role of intelligent human choices. Simulating card and dice games in this way is a great way to learn properties of the game itself that might be hard to recognize through play-testing or deduce by direct analysis of the rules. For example, as we will discover, the game LCR is *not* a fair game, but that can be hard to notice through casual play.

2 Software environment setup

Like the previous lab, you will be doing python programming. While you are allowed to prepare your python programs in any reasonable programming environment we continue to recommend PyCharm. The basic PyCharm setup instructions can be found in the previous lab. Some notes here:

1. You will want to make a new PyCharm Project for this new lab. Mixing labs in one project can lead to accidentally submitting the wrong file or other “carry-over” from lab to lab that might interfere with grading.
2. If PyCharm asks you to set a project interpreter, and no default shows up, you should be able to add one. On the CSELabs machines “/usr/bin/python3” is the name of the default python3 interpreter, which is a fine choice.
3. Remember to note where PyCharm makes the python project itself, you may even want to make sure the project is placed somewhere designated for this class when you set it up. Eventually you will need to find those files outside of PyCharm.

3 Lab Reminders

- Remember, labs are designed to be group work for groups of size 2

- If you do not have a lab partner and want one, there's no harm in asking, around, or in asking the TAs for help (they might know someone else looking for a lab partner)
- You are allowed to change lab partners each week if you want, you are never locked into one partner.
- You may want to try out working with a few different lab partners – It can help to see the programming style of several other programmers
- Try to use true pair programming – two programmers, one keyboard, one screen (not two people, each working independently). While one coder is typing, the other is actively looking for issues, looking up python syntax as needed, and planning ahead.
- Each programmer in a partnership should be equal – try not to run ahead of your partner, make sure they understand what you are doing, and make sure that each programmer spends equal time on the keyboard. (Remember, true pair programming is two programmers, one keyboard, one screen).

4 LCR Dice Game

The following information is based on the LCR rules found online at `https://www.wikihow.com/Play-LCR` and `https://www.dicegamedepot.com/lcr-dice-game-rules/`. You are free to review the game rules there as well.

LCR (short for "left right center") is a game largely focused around the movement of "coins" (poker chips, candy, money, whatever) around the players. Each turn the active player rolls dice to determine how to move their coins either keeping them, putting them in the center, or passing the left or right. The goal of the game is to be the only player with coins, at which point that player wins the coins they have and all coins in the center. The game is played with special 6-sided dice, three sides are labeled with a dot, representing that a coin can be kept, then one side each is labeled L (left), R (right) and C (center). It is from these dice that the game gets it's name.

At the beginning of the game, the players sit in a circle, each with three coins. One player is chosen to go first, afterwords play will go in a circle until the game completes.

A player's turn has three parts. First the player figures out how many dice they must roll. The player rolls as many dice as they have coins, capped at a maximum of three dice. So, if a player has three of fewer coins, they roll as many dice as they have coins, but if they have four or more coins, they would only roll three dice.

Then the player rolls their dice. For each dice that rolls an L the player must hand a coin to the player on their left. For each dice that rolls an R the player must hand a coin to the player on their right. For each dice that rolls a C the player must put a coin the center, coins in the center cannot be retrieved until the end of the game. If a dice rolls a dot, the player doesn't have to give up a coin.

If a player has no coins they are still in the game, but they roll no dice on their turn and nothing happens. (For our purposes, we will still count that as a turn, even though nothing happens) These player stay in the game because the player to their left or right may still be forced to give them coins.

The game ends when only one player has coins. This can happen on anyone's turn (so player 1 can be declared a winner at the beginning of player 3's turn, if no other player has coins).

There are many variants of this game, some of which add strategy (which I feel this game is sorely lacking, no one asked me). We will not be simulating those games, however, only the basic game.

5 Computational Representation of LCR Games

The major data that needs to be recorded about this game is how many coins each player has. The number of players is not fixed (it won't change as the game goes on, but we may want to simulate the game at different sizes so we can't just use one variable for each player), and the number of coins any player has will change. Therefore we will represent this count with a list of numbers, with each element of the list representing that player's coin count.

Likewise we will need to track whose turn it is currently. Since we are using a list to track coin counts, it is only natural to track players by their index. We will assume that player 0 always goes first (this is somewhat contrary to normal rules for the game). We will assume that "left" goes to the index one lower, and "right" goes to the index one higher, and that play goes to the right (so player/index 0 goes first, then player 1, then player 2 and so forth until it circles around to 0) Note that this assumption means for N players (numbered 0 though $N - 1$) player $N - 1$ is left of player 0 and player 0 is right of player $N - 1$.

We will represent the LCR dice with a function the returns "L", "C", "R", or "." with appropriate probabilities. This function is provided. Since we are not interested in the "winnings" of any given player, only who wins, we will not be recording the number of coins in the center.

This data representation is useful as it essentially reduces the game to a series of list manipulations, and allows us to track all information about the state of a game-in-progress with only two variables: the list of coin counts, and the current player's index.

6 Python syntax notes

A few reminders on python syntax. If you need more syntax help, the lecture slides, reading, TAs, and google can be great resources. If you find yourself needing to reference this regularly I **STRONGLY** recommend trying to compile a "Quick reference" guide.

We covered creating lists in class, but didn't get to talking about how to make lists of a given size. You can do this with a for loop by repeatedly inserting into the end of a list, but that's a lot of typing for something so simple. If you have a variable num_players and with to make a list full of 0s with num_players 0s in it, the following syntax will work fine:

```
coins = [0] * num_players
```

Note, this multiplication trick won't always work for initializing lists, but it works well enough for lists of 0. (you can substitute any number for 0 in this syntax without issue).

You can index lists to access specific elements either to read the current value or write a new value: `coins[player] = 3`

You can get the size of a list as **len**(coins).

If you want to loop over the elements of a list you can simply use the list after "in" in a for loop (I.E. **for** i **in** lst:) however, if you want to loop over the *indexes* of a list you will need to use the range function. You will also likely want to use the modulus operator(%) when determining the player left and right of the current player. I'll leave it up to you to experiment with how that might help.

I encourage you to take a minute as necessary to try out some of this syntax directly in the python interpreter before programming – you don't want to be typing syntax you arn't sure of. To launch Python in "interactive mode" (so you can try out syntax) open a terminal and type "python3".

7 Provided files

The following files are provided:

- `lcr_die.py` this module contains code for one important function (roll_die ()) which you will use to simulate LCR dice. It also has helpful code which we use in testing to help make sure the dice are being used as we expect (you should call the roll_die () function exactly as many times as the player would roll the dice, no more.)

- `lcr_test.py` this function contains tests for the required functions, and will serve as the basis of our grading.

8 Requirements and software design

While I would love to leave you at this point, give you design freedom, and see what you build, that would be better suited as a Project than a Lab. Therefore, (and for the sake of testing) I've gone ahead and reduced this application to a series of simple functions. All the functions listed here are required, but you are free to write other useful functions. Each of these functions has at least two tests in the testing file provided on canvas. To use that test file you MUST name your python module `lcr_sim.py`

- lcr_over (coins) this function takes an array of numbers (representing coin counts) and indicates if the game is over (that is, it returns true if only one element is non-zero, and false if more than one element is non-zero)

- lcr_winner(coins) this function takes an array of numbers (representing coin counts). You can assume that this function is only called when there is a winner, that is, this function will only be called if lcr_over (coins) returns true. This function should return the index of the player who won the game. That is, this function should return the index of the non-zero element of the coins list.

- left (coins, player) this function takes two parameters, the coins list, and the index of the current player. It should perform a "left" on that player's coin count, reducing it by one, and increasing the player to the left's coin count by one. Remember, as players sit in a circle, left of the "first" player is the "last" player. This function should return nothing, it should instead directly change the list.
- right(coins, player) like left, but performing a right shift, so removing one coin from player and giving it to the player to the right. This function should return nothing.
- center(coins, player) like left and right, but with the coin going to the center As we don't track coins in the center, this just needs to reduce the coin count for player by one.
- lcr_game(num_players) this function should simulate, to completion, a game of lcr. It should start with each player having 3 coins, and player 0 going first, and it should continue until only one player has coins per the rules of the game discussed above. This function should return a tuple with two values, the first value in the tuple is the index of the player who won. The second value should be the number of turns it took to win.
- most_common_winner(num_players) This function should simulate 2000 games with the given number of players. It should count how many times each player wins, and then figure out which player won the most over the 2000 games. This function should return a tuple with two values. The first value is the player who won the most of the simulated 2000 games. The second value is the percent of those games the player won (so win_count/2000). This function is the "end goal" of our program, it will tell us if the game is fair, and if not, how often a given player wins. This function should run relatively quickly (maximum one or two seconds).

9 Deliverable

Before submitting your work, please use the tests to carefully check that your code is bug-free. While one of the tests is subject to random chance, it is highly likely that your result will fall within the bounds listed.

Name your submission file `lcr_sim.py` and make sure that a comment at the top has your name, and if you worked with a partner, your partner's name. **If you worked without a partner and their name is not on your file they will not get credit, OR, you will be accused of cheating** (depending on if they turn in the file or not)

You should only submit the `lcr_sim.py` file. You will submit this through canvas. If you worked with a lab partner, only one copy of the file needs to be submitted, but both students' names should be in the file. If you worked with a lab partner, please make sure both you and your partner have a copy of the lab files for future reference.

This will be due before the beginning of your next lab next week. The exact time will be based on the official start time of your lab. Canvas will accept re-submissions, and will also accept late submissions. Please be sure you do not turn in the assignment late on accident.

If you finish this lab during the lab time, feel free to notify your lab TAs, and then leave early. If you do not finish this lab during the lab time you are responsible for finishing before the next lab.

10 More fun

What? Work *after* the deliverable? How could there possibly be more?

Of course there's more. This section is optional and not for credit but is offered as a guidance for how to take your project and further practice programming with it, or further explore the LCR game.

1. Write a function that outputs the average number of turns needed for the game to end
2. Write a function that computes the 90th percentile of turn lengths, that is, the number of turns which only 10% of games reach. Games longer than this can be seen as pretty uncommon.
3. Write a program that outputs tables of information varying for numbers of players, use this to answer the following question: is the game more fair or less as we add more players
4. How does the number of turns needed for the game change as the number of players changes? Can you find a way to plot this? (you don't have to plot this in python, but you can.)
5. Modify the function to allow different starting coins and max dice numbers. How does this effect the fairness of the game?
6. Look up variants of this game and try to incorporate them into your code.
7. Use this code to perform an analysis: are longer games of LCR more fair, or shorter? Explain your reasoning and methods.
8. Write a short essay about the structural failings of LCR
9. Use your knowledge to win a game of LCR.