Zack Traczyk
ztraczyk@ucsc.edu
Professor Long
03/30/21

CSE13S Spring 2021
Assignment 1 - Left, Right, and Center

DESIGN DOCUMENT

OBJECTIVE:

The objective of this program is to simulate a simple game called Left, Right, and Center. The game involves $1 < k \leq 14$ players who all start with $3 dollars. The players go around a circle rolling as many dice as dollars they hold (up to three dice). However, the dice are not 1-6, but are labeled with 3 x •, 1 x **L**, 1 x **R,** 1 x **C**. If a player rolls a • then they do nothing, if they roll an **L** they give a dollar to the person to the left, **R** the person to the right, and if they roll a **C** they put a dollar in the pot. The game is played until only one player has money left and that player wins the pot.

GIVEN:

- Faces enumeration (`PASS`, `LEFT`, `RIGHT`, `CENTER`)
- Die array of faces (3x `PASS`, 1x `LEFT`, 1x `RIGHT`, 1x `CENTER`)
- Array of character arrays containing names of 14 philosophers
    - Included as a header file
- Helper functions for getting the position to the right and to the left (aptly named `left` and `right`)

DATA TYPES:

There are a few things that need to be kept track of during gameplay:

1. Player a. Name b. Balance	To keep track of the player, only the balance needs to be stored. Since the index of the balance can match the index of the player name in the philosopher array only an array of balances needs to be stored as an unsigned integer.
2. Number/ Order of players	Use unsigned integers for the number of players total and currently in.
3. Pot	Use an unsigned integer.

APPROACH:

Lately I have been doing a lot of programming in Python where I have been using object oriented programming to break my programs into little pieces. With this project I was tempted to start doing this, however I quickly realized that the game did not call for this. Creating smaller functions and breaking the game states into different parts does not make sense and would only make the code longer and convoluted.

The game is short enough to fit the majority of the logic into main. However, one logical function that would help is *`roll`*. This function takes in an argument *`n`* and returns a random integer from 0 - `n` not including `n`.

PSEUDOCODE FOR MAIN:

```
// Ask for input
seed = ? input ?
randomseed(seed)
IF seed <= 0:
    ERROR

numplayers = ? input ?
IF numplayers < 2 or numplayers > 14:
    ERROR

// Each player starts off with $3
players = []
FOR numplayers:
    players[i] = 3

// Simulate game
inplayers = 0
pot = 0
WHILE TRUE:
    FOR numplayers AND inplayers > 1:

        // Skip a player if they have no money
        IF player[i].money == 0:
            SKIP THIS PLAYER

        rolls = player[i].money OR NO MORE THAN 3 Times
        FOR rolls:
            action = roll(6)
```

```
            // Skip a roll if player rolls a pass
            IF roll == pass:
                SKIP THIS ROLL

            decrease current player balance

            IF roll == LEFT:
                increase left player balance
                IF left player balance is not 0 anymore:
                    increase inplayers
            ELSE IF roll == RIGHT:
                increase right player balance
                IF right player balance is not 0 anymore:
                    increase inplayers
            ELSE IF roll == CENTER:
                increase pot

            IF current player balance == 0:
                decrease inplayers

            IF inplayers == 1:
                GAME OVER

// Find winner
FOR numplayers:
    IF player[i] HAS AT LEAST A DOLLAR:
        player[i] WINS
```