Assignment 2: Making a Dreidel Game

James Contini
10.12.22

Dreidel.h will initialize:

A main game function
Dreidel spin function
Round playing function
A function to check the status of the game
A variable to count the rounds
A variable to hold the coins in the pot
An integer which holds the winner's name index
A boolean finished game status variable
And lastly, an array of the names

Dreidel.c's control flow:

The function play_game is called with parameters for number of players, coins per player and a pointer to the number of rounds. Next an eight by 1 int array is created and then populated (for n_players) with the amount of coins each player should start with. This array represents the bank of each player. After this the game is actually played in a do while loop by a function called *play_round.* It takes as parameters, array bank, the memory address of the global variable of the pot, the number of players in the game, and the memory address of the integer (but used as boolean) variable *fin* which is used as the loop's run condition. While *fin* is 0, rounds play and a global variable n_rounds increments before each call, when *fin* is 1 the loop breaks and *play_game* returns the integer *winner* which contains the index for the array of names.

play_round: the round simulating function

play_round takes the aforementioned four parameters. Inside of this function, a for loop immediately runs with an index that goes from zero to less than the number of players in the game. So long as a player's bank isn't zero, a switch case with the expression *spin_dreidel()* will return either *N*, *G*, *H*, or, *S*. Each case corresponds to one of the Hewbrew letters and each case acts on the pot and players coin bank accordingly. If *N* the player's bank is set to itself, if *G* the player gets their bank plus the pot and the pointer of the pot is zeroed, if *H* the player gets half the pot (rounding down) and the pot is set to *the pot minus the pot over 2*, and lastly if *S* the players gets one coin removed from their bank and the pot gets one coin. This case also has a special check in it. If the player's bank becomes negative one after the coin is removed, then a function *chk_fin* is run which returns true or false that all players except one of them have negative one coins. If this is true then the global variable *fin* is set to one and the do while loop condition is no longer satisfied.

chk_fin(): the game status function

chk_fin takes parameters, *players[]* which is the bank array, the number of players, and the address of the *winner* variable which will house the array index of the winner. First, *chk_fin()* creates a local variable to represent the number of players who still have coins, then it is given a pointer. Next a for loop will index each player's bank and a conditional statement will check to see if they have any coins. If they have more than negative one coins, the local variable is incremented and the pointer to the address *winner* is set to their index number. However if multiple users have coins then that local variable will be greater than one and the winner's index number will re-assigned multiple times. So to avoid a false winner being asserted, nothing happens to the variable *winner* until the number of players with coins drops to one. And in which case *winner* will have been appointed to that player's index.

spin_dreidel(): the dreidel simulator function

This function is very simple: a random number is generated using the *mtrand_int64()*. Then we set an integer variable to modulo four of the random number. Lastly a switch case returns a Hebrew character depending on its alphanumeric order in reference to 0, 1, 2, or 3.

Play-dreidel.c's control flow:

Using *getopt main()* will parse command line options and their arguments. Command arguments from *-p, -c, -s,* will be fed to *play_game()*. And *-s*'s seed will be processed using strtoul and then fed to *mtrand_seed()*. The presence of *-v* will make a boolean value true to signify to *play_round()* that a printing elimination must occur. *play_game()* will return an integer, the index of the winner's name for the name array. In a printf, using format specifiers, the name, the ending round and the number of coins, the number of players and the seed, will be written in so they print as declared by the assignment.

Dreidel.h:

This file contains many extern global variables that can be changed and accessed from multiple c files. It also contains many global functions and two arrays.

Write-up scripts:

Monte Carlo simulation:

For my Monte Carlo simulation to find what the average length of a game is with 6 players and 4 coins I used a for loop to generate seed number and parsed the round number using modulo because play-dreidel's pattern is regular. Then I sent the rounds to a .dat file which was fed to GNUplot.

The second topic of the write up was also an easy bash script. I did the same thing except I also changed the player number seven times for each amount of players.

The last one was more difficult. The first thing I did to investigate if order position was relevant to winning was choose an environment. For this one I chose four coins and plotted a new graph for each

added player. I graphed the number of wins on the y axis against the player's number on the x axis. The player's number was determined by their lexicographical order.

For a plot with only Aahron and Betsheva:

The code loops through the names of the 4 coin 2 player game (Aharon and Batsheva) and would increment a variable which corresponded to the name when the name passed through the loop variable. Then redirecting to gnuplot, the number of wins for each player was plotted against themselves in box plot format. This boxplot will illustrate how many times a won and b won. What's great though is that the order of a and b is always constant so I can count on the order being the same each time. I will scale this up so I eventually plot a graph of wins against names in their play order for games with players 2-8.