# Design of Dreidel Game

Moore Macauley
University of California, Santa Cruz

October 16, 2022

## 1 Introduction

In this assignment, I was tasked to simulate a Dreidel game being played. Dreidel is played by a number of players, at least 2, who are all given the same number of coins. They each take turns spinning the four sided top, and depending on the result, they will do different things. First, if the result is nun/N, they do nothing. If they get gimel/G, they take the entire pot of coins. Given hayh/H, they only take half the pot, rounding down. Finally, if they roll a shin/S, they put a coin into the pot instead. Players are eliminated when they run out of coins, and the game ends when all of the players except 1 are eliminated.

In order to do this, I will create two c files, a header file, and use one provided c file and corresponding header file. The provided files are mtrand.h and mtrand.c, which use a Mersenne Twister to generate a series of pseudo random numbers given a seed. This seed must be greater than 0, and no longer than 10 decimal digits long. This program will be used to generate random numbers to determine the outcome of Dreidel rolls.

## 2 play-dreidel.c

The first file made by me, play-dreidel.c, will contain main() and will simply parse arguments from the command line and interpret return values from the second c file. It should accept the arguments -p, -c, -s, and -v. -p (int) corresponds to the number of players the game will be played with, min of 2, max of 8, default of 4. -c (int) is the number of coins each player starts with, min of 1, max of 20, default of 3. -s (uint64_t), and will be the seed used to generate random numbers to determine Dreidel spins. Finally, if -v is passed, the program should output a message whenever a player reaches 0 coins and is eliminated. If -v was included, it will be saved in an int that will be added to the dreidel header file, so it can be accessed by the second c file. It will then parse this information before passing it and a pointer to the number of rounds to the second c file by running the play_game() function, which will return a number corresponding to the player that won. In order to parse this return value, play-dreidel will contain an array with the player's names in it, with the value returned. It then prints out the name of the winning player, the number of players, the number of rounds it took, and the seed used.

```
Pseudocode:

Include dreidel.h, stdio.h, stlib.h, mtrand.h, and unistd.
Set player_names (defined in dreidel.h) to Aharon, Batsheva,
```

```
Chanah, David, Ephraim, Faige, Gamaliel, and Hannah

Create the variables players = 3, coins = 3, seed = 613, and a
pointer pointing towards round number = 0
Define the pointer message as pointing towards 0
Create an array of pointers using the example in figure 1
to create an array of the player's names, called player's names

Use getopt to iterate over the provided arguments
Switch option:
    Case 'p'
        Check to see if atoi(optarg) is between 2 and 8
            players = atoi(optarg)
        Otherwise
            return 1
        Break
    Case 'c'
        Check to see if atoi(optarg) is between 1 and 20
            coins = atoi(optarg)
        Otherwise
            return 1
        Break
    Case 's'
        Check to see if strtoul(optarg) is less than
        9999999999 and is greater than 0
            seed = strtoul(optarg)
        Otherwise
            return 1
        Break
    Case 'v'
        Set the value being pointed towards in message to 1
        Break
    Default
        Return 1
Set seed in mtrand to seed
Set outcome to play_game(players, coins, round number) (see below for info)
Print player's names[outcome], players,
the value pointed to be round number, and the seed

Return 0
```

## 3 dreidel.c

This other c file, dreidel.c, will contain 2 functions. The first, spin_dreidel(), will use mtrand.c to determine which side the dreidel lands on, and then returns a char corresponding to that. The second function, play_game(), will accept an int telling it the number of players, an int with the starting number of coins, and a pointer pointing to an int so it can return the number of rounds played. Using this, it will simulate an entire game of Dreidel from start to finish, before returning a number corresponding to the player that won.

```
Include dreidel.h, mtrand.h, and stdio.h

Pseudocode for spin_dreidel:

Generate a random number with mtrand.c, use modulo 4 on it, and save that to value

If value is 0, return G

If value is 1, return H

If value is 2, return N

Else, return S
```

To set up the game, my program will operate with an array. This int array will contain a number of coins the player in the same position in play-dreidle.c has. As an example, if the player array has Chanah in position 2, and the coin array has 8 in position 2, that means Chanah has 8 coins. This coin array will size 8, and will be initialized with the starting number of coins in every slot. I will also have an int variable for the number of coins currently in the pot, set to 0 to start, and an int that represents the number of players in the game, initially set to be the same as the number of players.

When actually playing the game, the function will iterate over a while loop, so long as the number of players in the game is greater than one. For each iteration of the while loop, I will iterate between 0 and the number of players passed into play_game(). It will then check to see if there are any coins in the coin array in the position corresponding to the iterating value. If there are none, then the player is out, and the loop moves onto the next player. Otherwise, it will spin a dreidel, using a switch to determine what to do with the result. If a G was spun, the number of coins in the pot is added to the player's coin count, before setting the pot to 0. If it got a H, the pot will be divided by 2, adding the result into the player's coin count, and subtracting the result from the pot. Thanks to integer division, the value will automatically be truncated and rounded down. Finally, if the player rolls an S, 1 will be added from their coin count and added to the pot. The program will then check to see if the player has any coins left. If they do not, the number of players in the game will be decreased by 1, and if the variable that determines if v was called is true, it will also print that the player was eliminated, and at what round. It then checks to see if the number of players left in the game is greater than 1, breaking out of the for loop if it is not. If no valid case is provided (which includes N), it does nothing. Outside of the for loop, the round count will be increased by 1.

Once the loop ends, the coin count will be iterated over one last time, checking to see which slot the one player that has coins left occupies. It will then return that value.

```
Pseudocode for play_game, includes the same headers as spin_dreidel:

Gets players, coins, and a pointer rounds pointing towards an int.

Set the value pointed to by rounds as 0.
Set players in game to players.
Set players temp to players
Create coin array of length 8, and initialize it with all slots
having coins number of coins
Create and set int pot to 0

While players in game is greater than 1
    Increase the value pointed to by rounds by 1
    For every number i between 0 and players temp,
        If coin array [i] > -1
            Use spin_dreidel and set it to result
            Switch result
            Case 'G'
                coin array[i] += pot
                pot = 0
                Break
            Case 'H'
                removed = pot / 2
                coin array[i] += removed
                pot -= removed
                Break
            Case 'S'
                coin array[i] -= 1
                If coin array[i] == -1
                    Then players in game -= 1
                    If the value in message == 1
                        Print that player's name[i] was eliminated
                    If players in game <= 1
                        players temp = 0
                Else
                    pot +=1
                Break
            Default
                Break

For every number i between 0 and players
    If coin array[i] > -1
```

```
        Return i

Return -1
```

## 4 dreidel.h

dreidel.h should contain all of the functions and variables that need to be passed between dreidel.c and play-dreidel.c. This should only contain a variable to check if v was passed, the play_game() function, and the array containing all of the player's names.

```
Pseudocode for dreidel.h:

paragma once

char * player names[8]
int *message
int play_game(int n_players, int coins_per_player, int * n_rounds)
```

## 5 Creating the Graphs in Writeup

For my writeup, I need to answer three questions, and to make this easier, I will generate histograms using gnuplot. The first question asks, how long does a game of dreidel with 6 players and 4 coins last? What is the longest game, and what is the shortest game? While the questions on the longest and shortest games can be answered logically, a graph would be helpful to answer how long a game lasts on average. To make this graph, I will edit my main function slightly, making it so that it only prints out the number of rounds a game takes. I will then use a for loop to iterate over my function, running the dreidel simulation for all seeds from 1 to 10,000, saving the results to a file. Once this is done, I can sort the results, getting the longest game that happened by finding the last line of that file, before counting the number of times each game length occurred. I then iterate in a for loop between 0 and the longest game that happened, saving the results in a different file that will be plotted. This will create a graph of how many times each length of game occured

```
Pseudocode for graphing.sh

For all values i between 0 and 10,000
    Save ./dreidel-test -p 6 -c 4 -s i into file results

Sort results
Save last line of results into max value
```

```
Save unique results and results that repeat into results that appear
Sort results that appear
Count the number of times each result appears, and save that into occurrences

Iterate over all values i between 0 and max value
    If i is in the first line of results that appear
        Save the first 7 bytes of occurrences into graph file,
        and add a newline
        Remove the first line of occurrences
        Remove the first line of results that appear
    Else
        Save a 0 into graph file

Plot a histogram with gnu plot with the file graph file
```

The second question asks if more players results in a game lasting more or fewer rounds. To graph this, I use the above script, changing the value of players that it feeds into play-dreidel from 2-8 and the name of the pdf the graph will be saved into.

The third and final question asks if position results in any particular advantage in a round. To graph this, I will once again edit play-dreidel a little so that it only prints out the position of the player that won. From there, I will perform the same method detailed in the first graphing script to create a graph of which players win the most often. Just to be safe, I will create graphs for this scenario for all possible numbers of players and coins. While these probably won't be used, I want to ensure that nothing changes when more players are added or more coins are used.