General Idea

- We are to write code that simulates a single game of dreidel.
  - We will use the Mersenne Twister code provided for us in asgn3-files.tgz.
  - The physical state will be the number of coins that each player has and the number of coins in the point.
    - We should note that the total number of coins in play is a constant of $p * c$, where p is the number of players and c is the number of coins each player started with.
    - The time step for this assignment will be a single player's turn.
    - The state after the turn depends only on the state before the turn and the random outcome of the spin of the dreidel.
  - Our simulation is played with up to 8 players.
    - If there are fewer than 8 players, we are to choose the first *p* name from the list.
    - Each player is given *c* coins to start.
  - The game will be played in alphabetical order of the players.
    - If a character has to put a coin into a pot and they cannot, they are out of the game.
    - The game continues until there is only one player left.
- The program will print out the name of the winning player along with the number of rounds it took to complete the game, where a round is one pass through all of the remaining players. It will also print out the random number seed.
- Our `main()` function calls `play_game()` to play a single game and will give us a number that corresponds to the player who won.
  - Our code in play-dreidel.c will translate that number to a string with the name of the player who won.
    - We *could* use a case statement to do this, but it's apparently much better to use an array of player names.
      - Define the array as a global variable and index into it when you need to print a name.
      - `nato_abc[1]` is a `char *` — a pointer — that points to the string (character array) `Baker`. Look at figure 2 for reference.
  - Include a function char `spin_dreidel(void)`, which will return one of the four sides of the dreidel.
    - We are to take in the return value from `uint64_tmt_rand(void)`, and choose the letter based on the result.
      - Assignment of letters must be alphabetical.
      - Your function must return `G` if the result is 0, `H` if the result is 1, and so on.
    - It should also contain an `include` guard, as discussed in class, either by using `#pragma` once or by using preprocessor variable and `#ifndef`.
  - We need to use `getopt()` to allow our code to take in parameters. For example, '`-c 4`' means that each player starts with 4 coins; your program must parse the number 4 from the command line arguments.
  - `-p` *n_players* : There are n_players players in the game (default: 4). (2-8 players (inclusive)). Use `int`.

        - `-c` *n_coins* : Each player starts with n_coins coins (default: 3). (1-20 coins (inclusive)). Use `int`.
        - `-s` *seed* : Seed the random number generator with seed (default: 613). Random numbers must be positive (non-zero) and must be no longer than 10 decimal digits. We will need `uint64_t` for your random number seed. Strongly suggest using the `strtoul` function from the Standard C Library.
        - `-v`: Print a message, shown in Figure 6, when a player is eliminated (default: no message).
        - Program must exit with a non-zero code.
- For our writeup, we must create a `bash` script that runs the appropriate Monte Carlo simulations, write the results to a file, and then process the file data to generate a plot with gnuplot. Or we could type out all the commands (which I might do).

<u>Pseudocode</u>

- If the user does not put a number within any of the ranges, we will return 1 as the exit code.
- Create an array of strings of the characters' names.
    - When the user inputs how many characters they want playing the game, the array will point to the amount they want from 0 to *n*.
        - Use `getopt()` and `atoi(optarg)` to get the amount of players.
        - Check that the input is valid.
        - For example, if they want 3 players, the pointer will first point to 0, then 1, then 2 during the game.
            - Create a variable to hold how many players there are, *n*, and if there is a need for a for-loop, always subtract one in the conditional to pull for the arrays.
    - The minimum number of players is 2, and 8 is the maximum.
    - Default case will be 4.
- Using `getopt()` and `atoi(optarg)`, we will get the amount of coins to give to each player.
    - Check that the input is valid.
        - If it is, create a for-loop that will assign that amount of coins to the characters' respective place in the coin array.
        - The other players who are not playing, if any, will have -1 coins. This is so that they are considered eliminated players and won't be able to play or move in the game at all, much like eliminated players.
    - Range will be 1 through 20.
    - Default is 3.
- Using `getopt()`, we will print a message when a player is eliminated.
    - To figure out if a player is eliminated, we will use an if statement to figure out if the character has 0 coins and lands on S.
        - If they do, give them -1 coins.
    - We will use `printf` and inside will have the pointers for the character's name and variables for the round and how many players.
    - Default will not print any message.
    - When a player is eliminated, subtract one from *n*.

- Since there are 4 faces on a dreidel, use an array and an index to find the face it will land on.
    - To "spin" (randomize it), we will use `char spin_dreidel(void)` and `uint64_tmt_rand(void)`. This will give us the *n* for what it will point to in the array.
        - If 0 is returned, that corresponds to G. Add to the character amount with the pot's amount. The pot will be 0 after. Add one to the indexing variable.
        - If 1 is returned, that corresponds to H. Take half from the pot, rounding down. Add to the character amount with the half's amount. The pot will be half after. Add one to the indexing variable.
        - If 2 is returned, that corresponds to N. Do nothing, but add one to the indexing variable.
        - If 3 is returned, that corresponds to S. Put a coin in if one can, if not, eliminate them. Add one to the indexing variable.
            - The variable that's correlated with the amount of players still valid in the game will get subtracted for each person eliminated.
- Create a while loop for the game to take in place.
    - There will be a variable that will hold how many rounds have taken place by adding 1 to it every time the game is able to reach a final state and it checks through that all conditions are still valid for gameplay.
        - Make an if-statement before adding a round to see if there is only one player left in the game. (Check *n*.)
        - Else: Print out the winner and other variables.
            - Create a for-loop to check which place in the coin array does not have -1 coins. The indexing number will be returned once it's found. It will be used to find the winner.
    - Set the while-loop's condition to continue if there's more than one player.
        - The game will only end if there's 1 player left, thus exiting the whole-loop.
    - The order of the game logic will be:
        - Go to the next player in the array.
        - Spin the dreidel and get the letter.
        - Do what the dreidel says using if-statements.
        - Check if there is only one valid player left.
            - If not, continue to the next person in the array.
- Put `main()` in play-dreidel.c. Call `play_game()`. Put a print statement below with the variables for what it needs to print out.
    - Get the seed using the given function.