Jadyn Fletcher & Samuel Graham Elmore

ITS 275

Dr. Munoz

April 8, 2020

ITS 275 Project Report

For our project, we proposed a simulator that would run games of Dreidel. The Dreidel game is a game of luck that consists of more than one player, a dreidel (a spinning top), and gelt (or other objects to bet with such as coins or candy), and a pot to hold the betting currency. A dreidel is a spinning top that has 4 sides, “nun” which requires nothing from the player, “gimel” which requires that the player receives all the contents of the pot, “hey” which means the player gets half of the pot’s content, and “shin” requiring the player to put their one of their betting currency in the pot. with each side requiring an action from the player who spins the dreidel. When the pot is empty, all players “ante up”, or submit one coin into the pot, and the round starts again. The game ends when one player possesses all the tokens of the game, while the other players have none. In making a simulator for the game of Dreidel, mathematicians will have better information to support them in their research of the game.

In developing our simulator, we initially planned on having one class that runs the logic and algorithms used to complete a game of dreidel. Once this was completed, we would soon move on to another class where we create the GUI. The GUI would include two text fields that the user could enter data into, one field for players and one field for the amount of gelt granted, in addition to having a submit button so the user could run the game with their desired parameters. In completing the schematic for the GUI, we decided that we would need to use exception handling that would block the user from submitting their data until the data entered by the user was an integer. Once the program was done running, we planned to have the information recorded by it be submitted to a txt file so that users could save the information to their records.

For the “Players”, this class holds the elements of the player and extends to the “GeltHolder” class. We created a “Player” object that would have our game keep track of players’ name with a unique identifier and the coins that the player had, in addition to managing their status in the game and a Boolean named “active” to hold that status. There are two constructors for the “Player” class, one without parameters automatically sets the status of the player to “true”, while the other takes an integer and assigns the gelt to the player along with setting their status to “true”. Every players’ name starts with “Player” in addition to having a number based on their position in the game. All the players are installed into an array as objects, and this array is made to be the size of players that the user requests through the GUI by a “for” loop in the “MainGame” class. As the game progresses, players’ coins are continuously received and redacted from the player’s gelt, and when their gelt is less than or equal to zero, the Boolean “active” is set to “false” through the “update” function in the “Player” class.

For the “Pot” class, this class extends to our abstract class “GeltHolder.” This class assigns the gelt and name from the abstract class, and also overrides the “write results” method of the “GeltHolder” class. When “writeResults” is called, \*\*add what we added\*\*

Our “MainGame” class is the class the at holds our logic, methods, and algorithms for the execution of the Dreidel Game. In the class, we assign the players (based on the number prompted by the user’s input in the GUI) to an array. These players are unique objects, and they all are assigned a name. The methods included in our “MainGame” class are: “spinPlayer”, “gameRound”, “anteUp”, “checkEndGame”, and “endgame.” The “spinPlayer” class begins with an int called “roll” that generates a random integer from 1 to 4 through the “Random” class. Proceeding this is a variable named “result” that holds the result of the Dreidel spin that’s done in switch. Based on the result of roll, the switch determines what the dreidel result will be. Following the switch that’s responsible for the result of roll, is another switch that determines the action the player takes based on the side the dreidel lands on. Concluding the method is a System.out.println that returns the name of the player’s dreidel result, the gelt that the player has, and the current contents of the pot.

The “anteUp” method begins with a for loop that goes through the “Player” array. Based on the active players in the array, the player is required to submit one gelt into the pot, which leads to the code deducting from the player’s gelt and adding 1 gelt to the pot. The final line of the “anteUp” method calls the update function of the “player” class to update the status of all of the players that accepts in the “for” loop based on the gelt that the player has after the ante up, in addition to the “checkEndGame” method. The “checkEndGame” method checks the number of remaining players, and if the number matches 1, then the “endGame” method is called. The “endGame” method \*\* goes through the “Players” array and checks for the only active player in the array, printing out a message that indicates which player won the game.

The “gameRound” method contains a “for” loop that goes through the “Players” array, making all the players run the “spinPlayer” method while also checking the gelt of the pot with an “if” statement. If the pot is empty, then the “anteUp” method is called, making all the players contribute one gelt to the pot. Concluding the “gameRound” method is a line that returns an ante up has been completed along with the contents of the pot.

One major implementation that was made upon the “MainGame” class is being able to export the information that’s run in the simulator to a txt file. The program does not print out the information from the “MainGame” class into the console since we took this route. The txt file is saved in a folder on the desktop by default by using the “createDirectories” method, and each file gets its own name. Files that are assigned to a collection of games are named after a single moment in order for them to be grouped together easily.

With the conclusion and success of our program, there are some updates & revisions we can make to better the experience of the program. One update that we could implement is including a button that lets you run one game at a time and assign that data to its own txt file instead only having the option of running and assigning them all to the same file simultaneously. In doing this, users can record their data according to their own preference. We could also implement an option to terminate the game when a specific player is out of the game. This can allow data collectors to discover the probability of a specific player clearer than if the game were to run until there’s only one player remaining.

The “alertBox” class holds the content needed create an alert in any other class. You can call the alertbox by typing “alertBox.display(Param1, Param2” with Param1 being a string for the title of the alert box, and “Param2” being a string for the message that the alert box prints. Regarding the contents of the “alertBox” class, one interesting line would be line 35, which deals with window modality. This line restricts the user from clicking on any other window from the application that launched the alert box, requiring the user to close the box before proceeding any further. The “alrtBox” class also creates a button that closes when it’s pressed.

The “Driedel\_Window” class is the final class that we have that includes the GUI. The GUI allows the user to input information in text fields for the number of players they’d like, the number of gelt that’s distributed in the game, and the number of games that are run, concluding with a button that allows the user to submit their data in. We use a method called “verifyInt” that handles exceptions through a try and catch loop to make sure that the data the user enters is in fact an integer, returning an alert box to let them know whether or not the information entered was valid or not.

UML Diagram:

