WCC – W19  
CS240  
Professor Ryan Parsons  
Adib Thafiq  
Andrew Jacobi  
Micah Miller  
Donald Strong

**Choices Game Project**

For our project, we were selected to create a game of choices that will ask the User a series of scenario-based questions designed to assess the value at which the User prioritizes their social, work, and school aspects of life. Over the last few weeks, we as a group have established a general foundation for the inner workings of the game and are currently working on bringing our initial ideas to life.

Our first order of business was the initial planning of how this game was to be structured. We decided that because the game will be a series of questions followed by two numerical values to represent a reward and punishment system, we thought it would be best to create these as Event objects. Next, we needed to figure out how we were going to separate these into the three aspects that we plan to assess (life, school, and work). We weighed the pro’s and con’s of using multiple text files vice a single text file, and are currently planning to use a single text file containing the data necessary to construct each event object.

The implementation of this, while seemingly elementary, proved that we had some rust to shake off. We created the Event object constructor which is designed to take in a String value – representing the question, a positive integer value (throws an exception if not positive) – representing the reward if the user answers “yes,” and a negative integer value that represents the punishment in the event that the user answers “no.” The constructor itself didn’t take long to write as it’s functionality is limited (for now). After writing the constructor, our next step was to create a “Manager” class that would be responsible for creating the objects with the provided input; this was where the rust really showed. The plan was originally to have the manager class read in three separate text files (one per aspect of life), though we hadn’t yet determined exactly how we were going to store them (partially due to our limited understanding of Hash Tables and their properties). After a brief run-down of Hash Tables provided by our instructor, we thought it be better if we stored all of the event objects in a single text file and give them a String “Tag” that will allow us to sort them into their respective aspects using a defined hashing algorithm. The creation of the manager class took longer than we had hoped, but we were finally able to create a working class that reads in a text file, creates a series of Event objects, and currently stores them in an arrayList which will soon be replaced with a Hash Table.

Of the two required data structures necessary for this project, we decided that our first one was going to be a Hash Table. Although at the time we had nearly zero understanding of how a Hash Table works at the time that we decided on implementing it, we still had a general idea of how we wanted this game to be structured that helped us in our determination. After doing some research (and admittedly feeling entirely clueless as to the differences between a hash table a hash map)- and a crash course from Professor Parsons- we decided to tag each event, which will be used to “sort” each event into a large array with 3 subarrays. Currently, the text files are designed with the following order:

Line 1: String representing the question

Line 2: Positive int value

Line 3: Negative int value

The manager loops through the entire text file and creates an object with the nextLine, followed by the nextInt, and finally the last nextInt before repeating the process for the next Event object. At the beginning of the String, each event will be categorized with the String starting with: “JOB,” “WORK,” or “CLASS.” Our current plan is to create an initial array of 60 indices (20 per aspect) to house each Event object. The manager will then take the length of the String tag, modulo it by the array size (which should yield three values) and place the object in its respective subarray (indices 0-19, 20-39, and 40-59). The hash algorithm will try to put each value into the same key (the first index in each subarray), but we will reserve the remaining indices in the subarray for overflow when handling the inevitable collisions. This is our preliminary line-of-thinking and is subject to change. Our timeline originally had us achieving our first implementation of a hash table by February 14th, 2019, though due to a series of factors, we may have to push this deadline back one week.

Our immediate goals are to finish our implementation of a hash table, and come together as a group to decide on our next course of action. We’ve been stressing the need to brainstorm how we plan to use a 2nd data structure for our program, and Adib and myself (Don) spoke briefly about possibly using a graph to implement some organization on the line of questioning during the game as opposed to the questions being seemingly random. This will allow a more sophisticated line of questioning that asks relevant follow-up questions depending on the user’s answer. Our group is meeting together on Thursday, February 14th, 2019 before class to hopefully iron out some of these long-overdue brainstorming activities and develop some additional concrete goals to achieve in the immediate future. I believe that the two next important tasks to accomplish are having a working hash table, and formalizing a decision on our next data structure and begin planning on how to implement it. I feel that these two tasks must be accomplished by February 21st, 2019 so that we can make up some of the time lost over the last week or so. Once we’ve completed this hurdle, we can sit down and write the code for our next data structure, then begin focusing on the main that will tie the whole game together. While we have yet to decide on a hard deadline to get an elementary version of our game working, I believe we will aim for sometime during the first week of March to ensure enough time for testing and adding some finishing touches.

At this time, we’ve accomplished some entry level testing to ensure that our Event constructor was working properly and that we were able to read in data from text files. We utilized the toString method defined in the constructor that prints out the Event.text along with the values for the reward and punishment. We’ve tried reading in valid text files as well as invalid text files (null values, values that should throw exceptions) and have yielded positive results so far. While we still have some progress to make, we’re currently on a good path and a realistic timeline to work with. As long as communication remains as strong as it has, and we continue making progress (despite the unforgiving weather), I believe that this project will be everything that we plan for it to be and more.