10/30/18

Project 1 Design

COMP-2710

1. **1st Use Case:**

Use Case: Initiate Game, created 10/23/18

Description: The player initiates the game by typing the number 3, which (when complete) will save the score and player username to a file. If more than ten scores/names have already been saved to the file, the save function performs a check against all existing scores and adds it in the appropriate location (assuming it is higher than at least one of the existing scores that are already in the file).

Actors: College Student

Assumptions: Score and username will properly be saved to the file in all foreseeable situations in which the user wins the game, scores will be appropriately calculated, individual attributes (intellect, time, money, steps) will not drop to a negative number, user should be able to quit at any time within the program, encounters/puzzles will have correct chances to appear, puzzles will have easy answers so an exact answer will not be necessary (will more than likely just request a single number), and a user will fail at appropriate times without a score being saved to the file.

Steps: 1.)  Player is prompted for their name for later usage (scoreboards). Happens before a case is selected from the main menu.

          2.) Player selects inputs the number 1 to start the game.

          3.) Player is assigned 20 intelligence points, 20 money points, 50 time points, and 20 steps. The steps will only change after a move has been completed and the value will steadily decrease by 1 point after each successful move.

          4.) Player is given 5 options: Move (initiates chances of certain encounters but can also avoid one altogether), Read Technical Papers (time – 1, intelligence + random number between 1 and 5), Search for Loose Change (time – 1, money + random number between 1 and 5), View Character (Displays all current attributes and how many steps are left to reach the end), and Quit the Game (quits immediately without saving score to file).

          5.) Random encounters will have chances of appearing during each move, which have the following chances/results: Nothing happens (10%, steps – 1, time - 1), Player steps into a warp portal through time (20%, time is either increased or decreased by a random number between 1 and 5 points, steps - 2), a Puzzle is encountered (30%, initiates puzzle (list of puzzles will be given in next step), steps – 1, other attributes vary with puzzle), a Professor is encountered (18%, time – 1, steps - 1, intelligence - random number between 1 & 5), another student requires cash for tutoring in an adjacent room (and you are a college student with money being offered, so you cannot say no) (20%, time – 1, steps – 1, money - random number between 1 & 5), or you suddenly become an overnight internet sensation and drop out of college to be rich and famous (1%, steps = 0, game is immediately won and score will be added to list if higher than all other scores).

          6.) During the event that a puzzle is triggered, each question will have the following chances/answers/results: “How many species of bats (out of the 1200 known species) are actually blind? (Enter a whole number between 0 and 9): “ (20%, 0, if correct: time + 1, intelligence + 1; if incorrect: time – 1, intelligence - 3), “How many deadly sins are there? (Enter a whole number between 0 and 9): “ (20%, 7, if correct: time + 1; if incorrect: time – 1, intelligence – 10), “What percentage of people in the world have an “outtie” belly button? (Enter a whole number between 0 and 9): “ (20%, 4, if correct: time + 1, intelligence + 2; if incorrect: time – 1, intelligence – 3), “How many bowel movements on average does a sloth have per week? (Enter a whole number between 0 and 9): “ (20%, 1, if correct: time + 1, intelligence + 5; if incorrect: time – 1, intelligence – 5), or “How many people have been to the bottom of the ocean (the Marianna Trench)? (Enter a whole number between 0 and 9): “ (20%, 3, if correct: time + 1, intelligence + 2; if incorrect: time – 1, intelligence – 3).

          7.) When a player lets any attribute become zero or below (except steps), they lose the game. A “You Lose” screen will be displayed upon a loss, and the results will not be added to the high scores list. When a player has their steps become 0 (it will never go below this number), their score is calculated by multiplying the intelligence, money, and time attribute values together. This score will then be compared with any others within the top 10 list (if there are already 10 scores entered in the list, this score value will be compared with all of the others, sorted within the list to be placed in the appropriate spot, and the user with the lowest score will be removed) and the results of the finished sorting will be written to a .txt file.

Variations: 1.) A player can win without stepping through every tile in the hallway.

                  2.) A player can win with a very low score and still make it onto the top 10 high scores list (assuming the game log is fresh or wiped before use).

                  3.) A person can lose either by running out of time, running out of intelligence, or running out of money. An appropriate losing screen will be displayed for each case.

Issues: Some functions may have to be created to handle incorrect type inputs (i.e. a string where an integer should be) so the program doesn’t immediately crash.

**2nd Use Case:**

Use Case: Top 10 High Scores, created 10/23/18

Description: Displays top 10 high scores that have been achieved by winning players. When a player wins, their score will be calculated and stored within a .txt file. This use case retrieves and displays said .txt file to the user when prompted with the number 2 as an input.

Actors: College Student, Any other player who has received a high score

Assumptions: 1.) There must be at least one high score entered into the file, otherwise an error message should be displayed upon attempting to utilize this use case.

                      2.) The list will be organized correctly and displayed in a neat, organized manner.

Steps: 1.) User must select the number 2 after being queried for their name.

          2.) The highest scores will be displayed on top of the list and the lowest scores will be on the bottom. Names of the winning players will be displayed beside the scores.

          3.) If less than 10 scores are available to display, the user will receive a statement after the last score to inform the user about the lack of high scores.

          4.) The menu will open again to allow the user to choose their desired decision for the game.

Issues: Finding a way to sort the list of winners appropriately each time while also removing any user who is no longer qualified.

**3rd Use Case:**

Use Case: Quit, created 10/23/18

Description: Closes the program when manually activated whilst also displaying a “You Lose” screen when necessary. Will choose appropriate losing screens for different situations.

Actors: College Student (User)

Assumptions: The Quit use case will have to perform in a multitude of areas and determine exactly what must be displayed in each situation in which it can be called.

Steps: 1.) User must select the number 3 after being queried for their name to immediately terminate the program if they do not wish to play the game, which will proceed to display a screen that simply states, “Come back when you aren’t afraid to discard 3 hours of your day”.

          2.) User must select the 5th option when prompted to move if they wish to quit the game early and forfeit a chance at being placed on the Top 10 High Scores list, which will display a general “You Lose” screen.

          3.) When the user loses due to 0 points on an attribute, the Quit use case will be called, which will display a screen that states, “You hit zero points on your \_\_\_\_ attribute! Try again if you think you’re good enough.”. If the user hits 0 points in the time attribute, it will display a similar message.

          4.) If the user wins the game, after the scores have been written to the Top 10 High Scores list, the quit function will be called and will display “You achieved a score of \_\_ points, congratulations! Check the Top 10 High Scores list to see if you made it!”.

Issues: Creating a versatile use case and programming it to have different outputs based on the parameters passed along to it.

**2.)** Menu, system, encounter, puzzle, highscores, character, quit

           i.) a.) Menu – Loads the menu when called so that the user can choose which of the 3 options they would like to use.

               b.) Member variables – decision, name; Functions: getDecision(), getName(), & mainMenu()

               c.) Does not depend on nor use another class to function.

               d.) Should receive one parameter within the constructor, which is the player’s name.

           ii.) a.) System – Instantiates objects and runs encounters to control the game.

                b.) Member variables: chance, name, steps, score, intelligence, money, time; Functions: getChance(), setChance(double chanceIn), getName(), getSteps(), setSteps(int stepsIn), runSystem(string nameIn), assignScore(), saveHighScores()

                c.) Uses input from Menu class; receives name to use within the system and pass along to other functions when necessary.

                d.) Should receive the user’s name as a parameter within the constructor. Sets initial character attribute values before passing them along to the Character class. Saves user name and score to file so that it can be accessed with the HighScores class.

           iii.) a.) Encounter – When activated, chooses between a list of available encounters with individual chances of being picked, then displays the necessary output for the chosen encounter.

                 b.) Member variables: chance, intelligence, money, time, steps; Function: encounterSystem().

                 c.) Receives input from System class, depends upon Character class to keep track of attributes.

                 d.) Should receive 4 inputs through constructor: int intelligenceIn, int moneyIn, int timeIn, & int stepsIn.

           iv.) a.) Puzzle – When activated within Encounter class, displays questions and prompts user for correct answer. Correct or incorrect answers will both cause the class to call the Character class to modify certain attributes.

                 b.) Member variables: intelligence, money, time, steps; Function: puzzleSystem().

                 c.) Depends upon Encounter class to receive attributes.

                 d.) Should receive 4 inputs through constructor: int intelligenceIn, int moneyIn, int timeIn, & int stepsIn. Will return either true or false when puzzleSystem() is used to determine whether the answer was correct or false.

           v.) a.) HighScores – Activated within Menu class if 2 is the input. Displays top 10 (or less) ordered scores to the screen and returns control back to the Menu class.

                b.) Member variables: string highScoreArray[10], string name[10], int score[10]; Functions: readHighScores() & displayHighScores().

                c.) Depends upon Main class to call this class and System class to write high scores to a file for access with this class.

                d.) Should have one constructor that accepts no parameters.

           vi.) a.) Character – Holds and modifies the character attribute values.

                 b.) Member variables: intelligence, money, time, steps; Functions: getIntelligence(), setIntelligence(int intelligenceIn), getMoney(), setMoney(int moneyIn), getTime(), setTime(int timeIn), getSteps(), setSteps(int stepsIn)

                 c.) Activated in every class, with Menu and HighScores being the exceptions. Depends on input to be passed down, modified, and then returned.

                 d.) Has one constructor that accepts 4 parameters for each attribute. Each parameter should be call by reference so that they can be properly modified.

           vii.) a.) Quit – Decides what to display when user loses, wins, or quits on purpose.

                  b.) Member variables: decision; Functions: quitSystem().

                  c.) Can be activated within Menu, System, Encounter, Puzzle, or Character if necessary.

                  d.) Requires one parameter when called, which notifies the class of which losing/winning screen to display on exit.

**3.)** Specific test cases will be made for HighScores, Character, Puzzle, Encounter, & System (may create ones for Menu and Quit later if necessary)

    i.) Character class

        a.) Will test to see if appropriate methods return the appropriate private values within this class.

        b.) Will test to see if appropriate methods return appropriate private values when negative (will be handled accordingly within System class).

        c.) Tests if the viewCharacter() method returns the appropriate string value to the user.

    ii.) Encounter Class

        a.) Each individual encounter method will be tested separately without the use of assert statements. They will mostly use if statements to allot for the random attribute changes that occur when called on.

        b.) The selector method will have the random value overridden if the constructor is called with a special “testInput” appended to the parameters. This will allow the specific output to be tested for each possible encounter by modifying the testInput. The puzzle encounter will only be tested to see if it can be selected from the list of available encounters; detailed testing of the puzzle class will be separate.

    iii.) Puzzle Class

        a.) Each individual puzzle method will be tested separately with the use of assert statements. We will be able to determine if the specific attribute changes have been made to the character class to determine if each method alters the character class attributes appropriately.

        b.) The selector method will have the random value overridden if the constructor is called with a special “testInput” appended to the parameters. This will allow the specific output to be tested for each possible puzzle by modifying the testInput.

    iv.) HighScores Class

        a.) The displayScores() method will be called with no scores and with some manually added ones to ensure the proper output is displayed. The tests for this method will be used with the aid of the other three methods.

        b.) The addScores() method will be called to add scores to the existing list of scores. This will be asserted with the help of the displayScores() method.

        c.) The sortScores() method will be used with the aid of the other three methods to determine if added scores can be sorted together and displayed properly.

        d.) The readScoreFile() method will be used with the aid of the addScores()  method to determine the correct amount of scores are read in.

    v.) System Class

        a.) The only method that will be tested is the runSystem() method. Every available action within the system will be tested individually (although encounter class will just be evaluated to see if it runs, as detailed testing of it occurs in a different test method). The tests will also evaluate each case of when the system can quit running, with a different one for each attribute dropping to zero. The points returned from the method (after forcing a winning run by overriding random chances within the encounter class) will be evaluated to see if it is correct.