Darrian Johnson
COMP 2710 – Ku
Project 1 Design
Monday, March 19, 2018

Project 1 Design Portion

Analysis

Start Menu: The start menu will appear following the user entering their name and a brief welcome message. From the start menu, the user will have the following choices:

- Start a new game
- View the Top 10 high scores
- Quit the program

If the user enters a valid command, then the action will be done. However, if the user fails to input a valid response, the game will prompt the user to try again. Once the user has successfully started and competed a new game, he/she will be prompted to try again. If the user selects "yes," they will be prompted with the start menu. Selecting "no," will end the program. This will continue until the user the option to 'quit' at the start menu or 'no' when prompted to try again. When the game concludes, the score will be determined by adding the time, intelligence, money, and subtracting the amount of moves left. Should the score be higher than the scores already in the list, it will be recorded into the list of high scores.

Turn Menu: If/when the user chooses to start the game, the user is given 5 options to choose from:

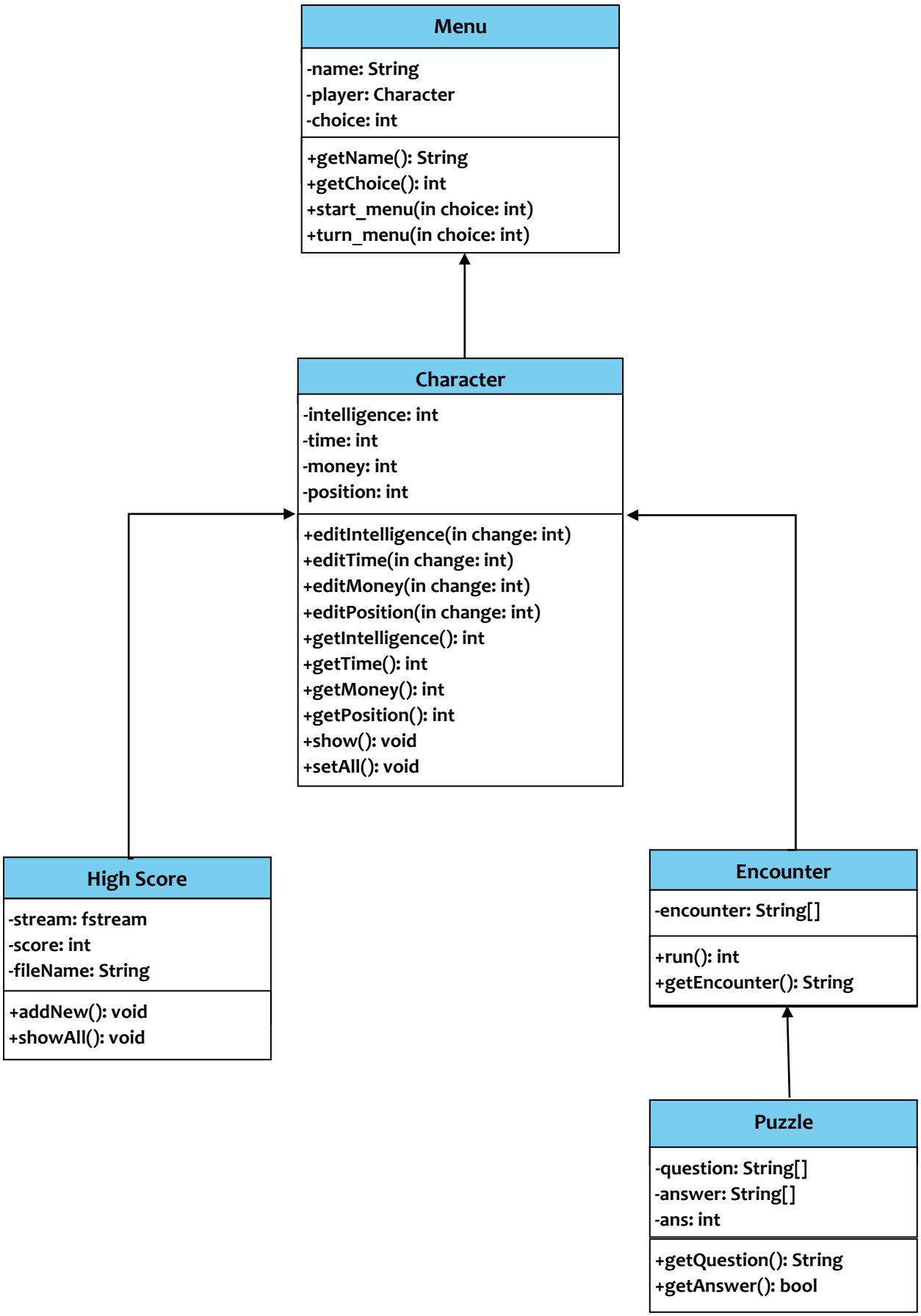
1. The user can choose to move one step on the grid and lose a value of time and risk and encounter.
2. Read technical papers to boost intelligence but lose a value of time.
3. Search for loose change to boost money but lose a value of time.
4. View the attributes and position of the character.
5. Quit the game.

When the user chooses to quit the game in the turn menu, the game will quit just as it would have if the quit option was chosen at the start menu. However, if the user chooses to quit the game in the turn menu, the score will not be recorded with the high scores. Every tenth turn, the user will have the option to do an exercise. The exercise allows the user to gain a move at the cost of them losing time.

Puzzle Choices: When the user encounters a puzzle, the user is prompted to answer a random question. The user is then given four choices of answers to choose from. If the user answers incorrectly, they will lose money. However, if the user answers correctly they lose nothing and gain a intelligence while also proceeding down the hall. If the user enters an answer that is not a choice, then he/she will be asked to try again.

Design

- **Menu:** The menu handles basic user interactions and errors such as choosing an option that is out of bounds.
    - Member variables – int choice, String name, Character player
    - Member functions – start_menu, turn_menu, getName, getChoice
- **Character:** This holds all the information pertaining to the character/player. This includes intelligence, time, and money of the character.
    - Member variables – int intelligence, int time, int money, int position
    - Member functions – editIntelligence, editTime, editMoney, editPosition, show, setAll, getIntelligence, getTime, getMoney, getPosition
    - Classes used/dependent upon: Menu
- **Encounter:** This provides the basis for an encounter. As the user gets closer to the end of the hall, the probability of a puzzle increases.
    - Member variables – String[] encounter
    - Member functions – run, getEncounter
    - Classes used/dependent upon: Menu, Character
- **High Score:** loads, sorts, and collects the high scores using a file input/output.
    - Member variables – fstream stream, int score, string fileName
    - Member functions - addNew, showAll
    - Classes used/dependent upon: Menu, Character
- **Puzzle:** This handles the encounter that are puzzles and the user interactions it requires
    - Member variables – String[] question, String[] answer, int ans
    - Member functions – getQuestion, getAnswer
    - Classes used/dependent upon: Menu, Character, Encounter


Menu
-name: String
-player: Character
-choice: int
+getName(): String
+getChoice(): int
+start_menu(in choice: int)
+turn_menu(in choice: int)
Character
-intelligence: int
-time: int
-money: int
-position: int
+editIntelligence(in change: int)
+editTime(in change: int)
+editMoney(in change: int)
+editPosition(in change: int)
+getIntelligence(): int
+getTime(): int
+getMoney(): int
+getPosition(): int
+show(): void
+setAll(): void
High Score
-stream: fstream
-score: int
-fileName: String
+addNew(): void
+showAll(): void
Encounter
-encounter: String[]
+run(): int
+getEncounter(): String
Puzzle
-question: String[]
-answer: String[]
-ans: int
+getQuestion(): String
+getAnswer(): bool

Testing

Test 1: This will test the functionality of the menus within the system. The program will start and ask the user for his/her name, then when they are prompted with the start menu that asks for a valid choice and the user enters a negative number, the player should be asked to try again until a valid form of input is given.

Test 2: This will test the programs ability to access high scores after entering a name. A correct input of two (**2**) in the start menu will be used to generate the viewing of the high scores. The program should display the players' name followed by their score in ascending order.

Test 3: This will test the turn menu which should appear each turn during the game. Once the user has successfully input a valid name and option of one (**1**) in the start menu they will be prompted to enter a number (**1-5**). Once the user inputs an invalid number of **-1**, they will be asked to try again.

Test 4: This will test the user's ability to access information about his/her character. After entering a name and a one (**1**) in the start menu, the user will have the option to access the character information from the menu that appears with every turn in the game. When the user enters a valid input of four (**4**) in the turn menu the information should display that the player has twenty (**20**) units of intelligence, twenty-five (**25**) units of time, and twenty dollars (**$20**) because these are the values that are set at the beginning of each game.

Test 5: This test will test the program's ability to detect if the user enters a valid name. Once the program starts and the user is prompted to enter a name, when the user enters the number five (**5**) the program should ask the user to try again since this is an invalid name. The program should not continue until a valid name has been entered by the user.