**Project 1 Final**

**\*\*All changes from original document are highlighted\*\***

**Compile information:**

To compile, have all files in the same folder. Use the following command:

g++ menu.cpp character.cpp system.cpp encounter.cpp puzzle.cpp game.cpp -o game

And then ./game to run, like normal.

If you wish to see/test each individual file, the file headers have compile information. Be sure to uncomment the main() function in them if you are going to do this. If you are just running the game, you can ignore this.

1. **Analysis: Use Cases**

* **Start Menu**:
  + **New game:** Starts a new game. Tells player starting attributes and brings up game menu.
  + **High scores:** Displays up to 10 high scores, then returns to main menu.
  + **Quit:** Exits the game
* **Game Menu:**
  + **Move:** move one space, lose one time. Chance of encounter/puzzle
  + **Read tech papers:** increase intelligence by random amount and lose fixed amount of time.
  + **Search for change:** increase money by random amount and lose fixed amount of time.
  + **View character:** displays character stats
  + **Quit:** Ends the game and exits the program.
* **Encounters:** encounters are randomly generated and have the possibility to affect any character stat.
* **Puzzle:** Similar to encounters. Are randomly generated and require the player to answer some sort of question or riddle.
* **Lose case:** The player loses if any stat drops to 0.
* **Win case:** The player wins if they make it to the end of the hallway without any stats dropping to 0.

1. **Design:**

* **Menu class:**
  + **Variables:**
    - **int playerChoice:** Holds the player’s menu choice.
  + **Functions:**
    - **(public) Menu():** Constructor, initializes playerChoice to 0.
    - **(public) int gameMenu():** Displays the game menu and returns playerChoice.
    - **(public) int mainMenu():** Displays the main menu and returns playerChoice.
    - **(private) void verifyGameMenuInput():** Ensures valid input for game menu.
    - **(private) void verifyMainMenuInput():** Ensures valid input for main menu.
  + **Class relation:** This class does not rely on any other class, although it is called by System.
  + **Other notes:** N/A
* **Character class:**
  + **Variables:**
    - **string name:** Name of the player
    - **int time:** How much time the player has
    - **int intelligence:** How much intelligence the player has
    - **double money:** How much money the player has
    - **int steps:** How many steps to reach the end of the hallway
    - **static const int DEFAULT\_TIME:** the default amount of time if not otherwise set.
    - **static const int DEFAULT\_INTELLIGENCE:** the default intelligence if not otherwise set.
    - **Static const double DEFAULT\_MONEY:** the default money if not otherwise set.
  + **Functions:**
    - **(public) Character():** Default constructor. Sets stats to default values.
    - **(public) Character(string charName):** Constructor. Custom name, other stats default.
    - **(public) Character(string charName, int timeIn, int intellIn, double moneyIn):** Constructer with all values entered.
    - **(public) void displayStats()**: Displays character stats.
  + **Class relation:** This class is called by System to create/manage the player. It is also called by Encounter and Puzzle to change stats based on the event.
  + **Other notes:** I really didn’t want to make the character stats public but having so many getters and setters seemed excessive. After learning about friend classes, I would go back and make the system class a friend class so it the stats can be private but still easily accessible by system.
* **PuzzleControl class:**
  + **Variables:**
    - **int playerAnswer:** The player’s answer
    - **Character player:** The player.
    - **Static const int NUMBER\_OF\_PUZZLES:** the number of puzzles.
  + **Functions:**
    - **(public) PuzzleControl(Character playerIn):** Constructor, sets player and random seed.
    - **(public) PuzzleControl():** Constructor, sets random seed.
    - **(public) void setCharacter(Character playerIn):** Sets the player
    - **(public) Character randomPuzzle():** Creates a random puzzle, runs it, then returns the updated character (stat changes).
    - **(private) double fRand(double fMin, double fMax):** Random function for double, allows easy random number generation for money gain/loss
    - **(private) Functions for each puzzle:** These functions are: shelbyOpen(), bigger(), farmer(), months(), pattern(), jimmy(). Each one displays a puzzle and asks for the answer, and then changes stats appropriately.
    - **(private) verifyInput():** verifies the input is valid.
  + **Class relation:** Is called by System, and calls Character class to change character attributes.
  + **Other notes:** N/A
* **EncounterControl class:**
  + **Variables:**
    - **Character player:** The player.
    - **Static const int NUMBER\_OF\_ENCOUNTERS:** the number of encounters.
  + **Functions:**
    - **(public) EncounterControl(Character playerIn):** Constructor, sets player and random seed.
    - **(public) EncounterControl():** Constructor, sets random seed.
    - **(public) void setCharacter(Character playerIn):** Sets the player
    - **(public) Character randomEncounter():** Creates a random encounter, runs it, then returns the updated character (stat changes).
    - **(private) double fRand(double fMin, double fMax):** Random function for double, allows easy random number generation for money gain/loss
    - **(private) Functions for each encounter:** These functions are: professor(), gradStudent(), papersToGrade(), banana(), ex(), and watch(). Each one displays an encounter and then changes stats appropriately.
  + **Class relation:** Is called by System, and calls Character class to change character attributes.
  + **Other notes:** N/A
* **System class:**
  + **Variables:**
    - **Static const int MAX\_SCORES:** The maximum number of scores to display
    - **Static const int FIXED\_TIME\_LOSS:** The time loss for collectinh change or reading papers
    - **Character player:** the player
    - **Menu menu:** the menu
    - **EncounterControl encounter:** controls encounters
    - **PuzzleControl puzzle:** controls puzzles
    - **Int numOfScores:** the current number of scores actually in score array
    - **String highScoreFile:** the file that holds the high scores.
    - **Highscore highscore\_array[]:** The array of high scores
    - **Bool start:** keeps track of if it is the start of a new game or not
    - **Bool name:** true if name is needed, false if not.
  + **Functions:**
    - **(public) System():** Constructor, initializes variables
    - **(public) void gameStep():** This is what controls the main game functionality. It displays the game menu, processes the player choices, and makes corresponding calls to other class and functions. It also checks the win/lose case.
    - **(public) void displayHighScore():** Displays up to 10 max scores.
    - **(public) void addHighScore(string name, int score):** Adds a highscore to the highscore array**.** Also adds it to the file.
    - **(public) Character getCharacter():** returns the character
    - **(public) void setScoreFile():** Sets the file the scores are written to. Mainly used for testing.
    - **(public) bool replay():** Asks the user if they would like to play again.
    - **(public) void readPapers():** Read papers, gain random intelligence.
    - **(public) void changeSearch():** search for change, gain random amount of money.
    - **(public) void endgame(bool win):** Runs end game functions like displaying score and asking to replay. Runs different operations if player has won (win = true) or if they have lost (win = false).
    - **(private) void fillScoreArrray():** Fills the score array with highscores from the score file
    - **(private) void emptyScoreArray():** clears the score array.
    - **(private) double fRand(double fMin, double fMax):** Allows easy rand() use for doubles, for money randomization
    - **(private) void writeHighScore():** writes the highscore to the score file
  + **Class relation:** This is the backbone of the program. It calls the other classes to put together a cohesive game.
  + **Other notes:**
* **Highscore Struct:** Structure that holds highscores. Two variables: string name, int score.
* **game.cpp:** Extra file that contains the main() function for the program. Calls the system functions to run the game.

1. **Test Cases**

* **Normal Usage:**
  1. Typing a number (1, 2, 3, 4) should call the corresponding menu option’s function.
  2. During a puzzle, the correct answer should give the positive outcome. Wrong should give negative.
  3. Any movement down the hallway should move one step closer and use one time unless the encounter changes these numbers.
  4. Winning the game should add their score to the high score list.
  5. Quit option for main menu and game menu should exit the program.
  6. Show scores option should display up to 10 high scores.
* **Abnormal usage:**
  1. Any invalid input should be caught and prohibited.

**A screenshot of a cell phone

Description generated with very high confidence5. Test Results**

**Screen of a cell phone

Description generated with high confidenceCharacter class Testing:**

**A screenshot of a social media post with text and a black background

Description generated with very high confidenceA screenshot of a cell phone

Description generated with very high confidenceMenu Testing:**

**System Testing**

A screenshot of a cell phone

Description generated with very high confidenceA black sign with white text

Description generated with very high confidenceA screenshot of a cell phone

Description generated with very high confidenceA screen shot of a social media post

Description generated with very high confidenceA screen shot of a social media post

Description generated with very high confidenceA close up of a screen

Description generated with very high confidence