Analysis: Use cases

* Display startup menu with three options
  + Option 1: Stars a new game. This will display the starting attributes for the player(generated randomly at start??)
  + Option 2: Display high scores. This will read from the high score file and display the top 5 scores.
  + Option 3: Exit game. This option will terminate the program.
* Game menu: Options before each turn
  + Move: This option moves the player forward one space and takes one second off of the time. Also, with each move comes the possibility of an encounter or puzzle.
  + Search for loose change: This will increase the player’s money by a random value, but it will also take off some time.
  + Read technical papers: The player’s intelligence will increase by a random value, and there will be a fixed amount of time that is taken off for this option.
  + View Character: This option should display the current player’s attributes along with their current position in the hall.
  + Quit: This will terminate the game, and will display to the output, “You lose….Loser.”
* Encounters: This case is a possibility after every move.
  + 15% nothing happens. Proceed with next move as normal.
  + 30% chance of encountering a puzzle.
    - Puzzle: This interaction will differ from the others due to including user input.
      * Q: David’s father has three sons: Snap, Crackle and \_\_\_?

A: David

* + - * Q: What’s a lifeguards favorite game?

A: Pool

* + - * Q: Why do hamburger’s fly south for the winter?

A: So they don’t freeze their buns

* + - * Q: What did the cheerleaders say to the ghost?

A: Show your spirit!

* + - * Q: What do you get when you have a cat that eats lemons?

A: A sour puss

* + - * Q: Why is a bad joke like a bad pencil?

A: Because it has no point

* + - * Q: How do you get a baby alien to sleep?

A: You rocket

* + 10% chance of encountering a professor. This will slightly increase intelligence, but will remove some time from the total.
  + 10% chance of encountering another student. This will remove time and money because you both need coffee.
  + 15% chance of getting swamped with homework. This takes off time and intelligence (brain dead).
  + 10% chance of having to grade programs. This will remove time, but it will add in some $$$$.
  + 10% chance of taking a nap. This will remove time, but add intelligence.

Design

* Menu: This object holds the game menu described earlier.
  + Variable(s): choice: This will be of type char. Holds the choice that the player choses.
  + GetInput() Function
    - This will be a function that will get the input from the user and assign it to the variable choice. This function returns the choice from the input.
* Game System: Controls most of the game play. This will call to other functions as necessary.
  + Variables:
    - int subTime; This will hold the amount of time to remove from the total time if the player comes to an encounter that requires a loss in time.
    - We will need to come up with a maximum and minimum starting values for time, intelligence, and money. These will be of type int for time and money, but of type double for money.
    - Int hall\_length. This variable will hold the length of the hallway. (Must be at least 20)
  + Functions:
    - void newGame(); This will instantiate variables, call GetInput function from menu, and call functions based on input.
    - void setSubTime(int time); This will set the amount of time lost when doing an action
    - void setTimeBounds(int min, int max); Sets min and max starting values of time
    - void setIntelBounds(int min, int max); Sets min and max starting values of intelligence
    - void setMoneyBounds(int min, int max); Sets min and max starting values of money
    - void setHallLength(int length); Sets length of hallway.
    - void readPapers(); Read papers, lose time, gain intelligence
    - void findChange(); Search for change, lose time, gain money
    - void showHighscore(); Prints highscore
    - void saveScore(); saves score
    - void endGame(); ends the game
  + Class Relation: Since this class does most of the heavy lifting for the game, it calls to every other class in the code.
* Encounter: This class/method will determine what ecounter will be selected when called upon.
  + Variables: We will make variables to represent the percentages of each individual encounter.
  + Functions:
    - Void generateEncounter()
      * This function will generate a random number and compare that number against the variables and their probabilities listed above.
    - Class relations: This class will call to the character class to change the current player’s stats if necessary. Also, if the encounter happens to be a Puzzle this class will call the puzzle class.
* Puzzle: This class will determine what puzzle will be selected when it is called upon by the Encounter class.
  + Variables: This class will probably not need any variables to itself. Will add in variables when necessary.
  + Functions: Public
    - Void randomPuzzle()
      * This method will generate a random puzzle using a random number generator to select an individual puzzle.
    - Class relations: The Puzzle Class will be called upon by the encounter class. The Puzzle class will also call to the Character class to change the player’s stats when necessary.
* Character: This class will hold the individual character’s stats/attributes.
  + Variables:
    - Int intelligence: This represents the player’s intelligence.
    - Int money: this holds the player’s amount of money.
    - Int time: this represents the time the player is given.
  + Functions: Public
    - Void printStats()
      * This function will display the current attributes that the player has.
    - Void setMoney(int Money\_in)
      * This function will set the initial amount of money.
    - Void setIntel(int intelligence\_in)
      * This function will set the initial amount of intelligence.
    - Void setTime(int time\_in)
      * This function sets the initial amount of time.
    - Int getMoney()
      * This function returns the current player’s money.
    - Int getTime()
      * This function returns the current player’s time.
    - Int getIntel()
      * This function returns the current player’s intelligence.
    - Void sub\_intel(int amount)
      * This function removes the amount of intelligence lost.
    - Void sub\_money(int amount)
      * This function removes the amount of money from the player’s total amount.
    - Void sub\_time(int amount)
      * This function removes the amount from the total player’s time.
    - Void add\_time(int amount)
      * This function adds time to the player’s overall time.
    - Void add\_money(int amount)
      * This function adds the variable amount to the player’s overall money.
    - Void add\_intel(int amount)
      * This function adds the variable amount to the player’s overall intelligence.
  + Class relations: This class will be called upon anytime a player’s stats change.

Testing Scenarios

* Normal Cases:
  + Typing a number should call the correct menu option
  + Verify correct outcome for answering puzzles correctly and incorrectly.
  + Movement down hall should decrement the length of hall and use one time unless outcome specifies otherwise
  + Quit for main menu should end program
  + Quit for game menu should return to the main menu.
  + The show scores option should show the top 10 scores.
* Abnormal Cases:
  + If max time is set less than hallway length, the user should be warned the game can be impossible.
  + The input should always be checked.
    - Any time the main menu or game asks the user for input, it should only accept viable input values. If the input is non-viable, the program should ask the user to re-enter their input in the required format.
    - The puzzles should also check for valid input.