COMP 2710 Software Construction
Spring 2018
Project #1

Dragons – A Simple Text-Based Game

Points Possible: 100 (30 design, 70 programming)
Due:
Design Portion: **October 30, 2018 by 11:59 pm** on Canvas
Programming Portion: **November 9, 2018 by 11:59 pm** on Canvas

Goals:
To gain proficiency with basic C++ syntax.
To organize code into classes.
To gain familiarity with formal testing.
To write a fun application!

Design Portion:
Create a Word or PDF file named "Project1 Design". Please submit a Word (.doc) or PDF (.pdf) file.

Process (Steps 1-3 due with design portion, Steps 4-5 due with programming portion):
Each of the following should address the program specification below. Concern yourself more with thinking about the problem than worrying about the specifics of implementation at this stage. For the *design portion* turn-in, you will provide Steps 1-3. For the *programming portion* (final) turn-in, you will provide all the steps (including any revisions you may have made). Please note revisions in your final copy. You should provide:

1. **Analysis (10 pt):** Write a few important **use cases** (at least three). Remember, these describe how a user interacts with the system (what they do, what the system does in response, etc.). These need not be long, but they should have enough basic details such that someone unfamiliar with the system can have an understanding of what is happening. They should not include internal technical details that the user is not (and should not) be aware of. Make sure that any special rules/features you plan to add are clearly described in your Analysis section.

2. **Design (10 pt)**: From your use cases, identify likely candidates for software objects. List the classes (or draw a Class Diagram if you like) and include:

1) The name and purpose of the class
2) The member variables and the functions of the class
3) Any other classes that this class depends on or uses
4) Any relevant notes that do not fit into the previous categories

3. **Testing (10 pt)**: Develop lists of specific test cases (at least five):
For the system at large. In other words, describe inputs for "nominal" usage. You may need several scenarios. Also, suggest scenarios for abnormal usage and show what your program should do (for example, entering a negative number for a menu might ask the user to try again).

Remember, test cases are specific, repeatable, and should have a clearly defined result. Testing functions with randomization may require specialized drivers.

4. **Implement your solution in C++** (65 pt).

5. **Test Results (5 pt)**: ***After developing your solution***, actually try all of your test cases. Actually show the results of your testing (a copy and paste from your program output is fine – do not stress too much about formatting as long as the results are clear). You should have test results for every test case you described. If your system does not behave as intended, you should note this.

Program Portion:
The customer wants a cool new video game, similar in style to popular video games today. However, the customer recognizes the lack of significant funds and only has a resource poor computer to play this game on. Hence, the game will be a simple text-based adventure concerning a graduate student trying to navigate his way down the Shelby Center. The game has properties as follows:

- The "player" is represented by *at least* three attributes: *intelligence*, *time*, and *money*. If the player runs out of intelligence, time or money, the player dies. The goal of the player is to survive to the end of the "hall" with the highest combined total of the attributes as possible. "Score" is determined as the three attributes multiplied together.
- The player starts the game at the beginning of a hall, which is linear.
- The "Hall" is a path that is at least twenty (20) steps long. After a move, the user should be told how far away from the goal they are (in steps). If the player survives to the goal square without any of the attributes falling to 0, they win. Their score should be displayed with a simple victory message. If the player dies, a "You Lose" message should appear indicating the cause of death (for example, if money falls to zero, you can say that the player starved to death because of poverty). All the attributes should start in some random range (e.g., 8-25).
- Every turn, the player has (at least) 5 options to choose from:
    - **Move**: The player moves one step in the grid, but risks an Encounter or a Puzzle. Moving also takes time.
    - **Read technical papers** (boost intelligence): The player loses a fixed amount of time, but increases intelligence by a random amount.
    - **Search for loose change** (boost money): The player loses a fixed amount of time, but increases money by a random amount.
    - **View character**: A simple display should show the character attributes and current position in the hall.

    - **Quit the game** (shows the "You Lose" screen (optional mockery) and exits the program).
- Encounters: Every time the character steps, there is a random chance of various events happening. You are free to change the probabilities as you see fit for "game balance," but here are some suggestions:
    - 25% chance: nothing happens, you just move forward.
    - 30% chance: You encounter a Puzzle (see Puzzle below)
    - 10% chance: Encounter a professor. This loses a random extra amount of time, but may slightly increase intelligence.
    - 10% chance: Encounter another graduate student. This loses a random amount of time.
    - 15% chance: Attacked by grunt work! Lose both time and intelligence.
    - 10% chance: Grade papers. Lose time, but gain money.
    - 0% chance: Get a huge raise, gain lots of money! (This never happens).
- Puzzles: Puzzles are different from normal encounters since they require interaction from the user. These do not necessarily have to be brilliant, but riddles or even edutainment light puzzles are fine. Examples:
    - "What is 2 + 2:" For a correct response, Money + 1. For an incorrect response, Money – 20 (you idiot).
    - "What can you put in a barrel to make it lighter?" For a correct response, int + 2. For an incorrect response, int – 2.
- You are free to add more details and rules to your game, but you must have at least these specifications. Feel free to be creative – there are many opportunities to do so.

Write a program called Project1.cpp.

Use comments to provide a heading at the top of your code containing your name, Auburn Userid, and filename. Also describe any help or sources that you used (as per the syllabus).

You will lose points if you do not use the specific program file name, or do not have a comment block on **EVERY** program you hand in.

The program first prompts the user for a name by printing "What's your name?", and then prints a welcome message to the user, which should be centered on the screen and surrounded by a box.

Then, the user should be given a menu of options, such as the following:

1) Start a New Game of Shelby Center and Dragons!
2) View Top 10 High Scores
3) Quit

Starting a new game will perform as described above. Viewing the top 10 high scores will require those scores to be read *from a file*. The scores need to be *sorted* from the highest to the lowest. If a game results in a score higher than the existing high scores, you should replace the lowest score with the new higher score. The user should always have the ability to quit and this should exit the program normally.

Your program's output need not exactly match the style of the sample output (see the end of this file for one example of sample output). However, please note that since the program is using lots of randomization you may not get the same results each time you test!

Important Notes:
- You must use an *object-oriented programming strategy* in order to solve this problem (in other words, you will need to write class definitions and use those classes, you cannot just throw everything in main()). A well-done implementation will produce a number of robust classes, many of which may be useful for future programs in this course and beyond. Remember good design practices discussed in class:
    a) A class should do one thing, and do it well
    b) Classes should NOT be highly coupled
    c) Classes should be highly cohesive

Some potential classes:

A Menu class which handles basic user screw-ups (choosing an option out of bounds).
A System class which instantiates the other objects and runs encounters. Receives input from the Menu class.
An Encounter class which provides the basic framework for a generic encounter (with additional information feed in from system as the encounters occur).
A Puzzle class/function which operates with Encounter.
A HighScores data type which loads, sorts, and collects the high scores.
A Character class which keeps track of all the data relating to the character.

You should at a very minimum have classes to handle menus, and deal with encounters

- For the high scores, you will have to use some simple file input/output. When you have to update the high scores, it is easier to just overwrite the scores file than to try to modify an existing scores file.

- You DO NOT need any graphical user interface for this simple, text-based game. If you want to implement a visualization of some sort, then that is extra.

Error-Checking:

You should provide enough error-checking that a moderately informed user will not crash your program. This should be discovered through your testing. Your prompts should still inform the user of what is expected of them, even if you have error-checking in place.

Submission

Please include all of your design artifacts (Steps 1 – 3) within a single Word or PDF file.

The programming portion should include all the C++ source files for the game. If your program consists of multiple source files, please compress them into a single zip file. In your submission, provide a detailed description of how to build your program. If the TA cannot compile your code, you will not get credit for this portion.

In addition, include a text file that contains the execution results of your test cases. Please name this file **Project1_test.txt**.

Submit your design document, program source files, and test results through Canvas.

All documents should be submitted in digital format; no hard copy is required for the final submission.

Sample Usage:

```
        What's your name? Bob


        ===========================================================
        |                    Welcome, Bob!                        |
        ===========================================================

   1) Start a New Game of Shelby Center and Dragons!
   2) View top 10 High Scores
   3) Quit

        Please choose an option: 2

   The top 5 High Scores are:

   Jack 1337
   CaseyZZZ 625
   Aubie 400
   Bob 75
   Daisy 33
   -no more scores to show-


   1) Start a New Game of Shelby Center and Dragons!
   2) View top 10 High Scores
   3) Quit


        Please choose an option: 1

Entering the Game…
```

```
You have:

intelligence: 20
time: 25
money: $11.00

You are 20 steps from the goal. Time left: 25.

         1) Move forward (takes time, could be risky…)
         2) Read technical papers (boost intelligence, takes time)
         3) Search for loose change (boost money, takes time)
         4) View character
         5) Quit the game


      Please choose an action: 4



You have:

intelligence: 20
time: 25
money: $11.00

You are 20 steps from the goal.  Time left: 25.

         1) Move forward (takes time, could be risky…)
         2) Read technical papers (boost intelligence, takes time)
         3) Search for loose change (boost money, takes time)
         4) View character
         5) Quit the game


      Please choose an action: 2

You read through some technical papers. You gain 3 intelligence, but
lose 2 units of time.

You are 20 steps from the goal.  Time left: 23.

         1) Move forward (takes time, could be risky…)
         2) Read technical papers (boost intelligence, takes time)
         3) Search for loose change (boost money, takes time)
         4) View character
         5) Quit the game


      Please choose an action: 1

You move forward one step, and…

NOTHING HAPPENS!
```

```
You spent one unit of time.

You are 19 steps from the goal.  Time left: 22.

         1) Move forward (takes time, could be risky…)
         2) Read technical papers (boost intelligence, takes time)
         3) Search for loose change (boost money, takes time)
         4) View character
         5) Quit the game


      Please choose an action: 1

You move forward one step, and…

YOU FIND SOME PAPERS TO GRADE.

You spent two units of time, but gained $3.00!

You are 18 steps from the goal.  Time left: 20.

         1) Move forward (takes time, could be risky…)
         2) Read technical papers (boost intelligence, takes time)
         3) Search for loose change (boost money, takes time)
         4) View character
         5) Quit the game


      Please choose an action: 1

You move forward one step, and…

PUZZLE: It’s a riddling imp. I hate riddling imps. But fine, he asks:
“Find the product of 8 and 8!”

   1) 16
   2) 64
   3) 256
   4) Uh…uh… no?

Choose wisely: 4

The imp cackles “Oh yes. Yes indeed. Now you die.”

TIME HAS FALLEN TO ZERO. YOU DIE.

<Print Score, adjust high scores>
```