Project 1

Title:

Dungeon Crawler

Student: Hassan Farhat

Submitted to Professor Mark Lehr

Due Date: 28th October, 2016

Course and Section:

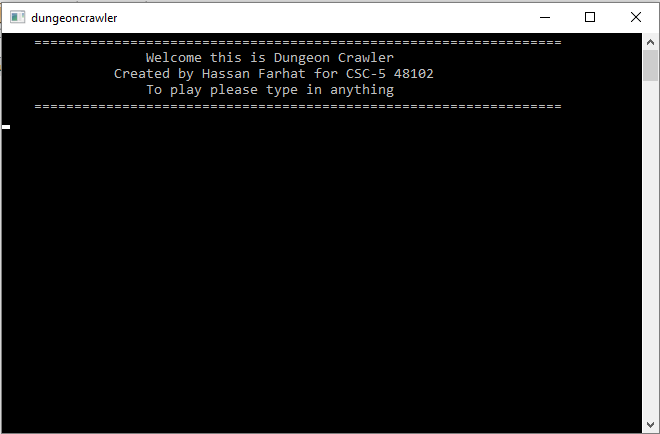
CSC-5, 48102

Table of Contents

1. Introduction
2. Gameplay
3. Code Content:
   1. Number of variables and number of lines.
   2. Table of concepts used and where to find an example.
4. Code Break Up: (Each of these will include Pseudocode, some will include a small flowchart, and most will contain sample code from the project.)
   1. Skill Set Up.
   2. Shop.
   3. Items.
   4. Movement and map.
   5. Spawn and decision to engage.
   6. Battles.
   7. Scoreboard.
5. Pseudocode Representation of the game.
6. Project 2 plans.
7. The full code.
8. **Introduction:**

The adventure game has been around for many years. There are many type of adventure games RPG, MMORPG, JRPG, board games (Dungeons and Dragons) and many more. One of the most prominent features these games have are dungeons. Dungeons are complexes made of many rooms that may contain enemies, and the final room contains a boss. The act of fighting you way through these dungeons is called dungeon crawling.

Dungeon crawling is such a popular concept that many games are solely dungeon crawlers. Many of the first text adventures were dungeon crawlers, and the 1980s is considered the “Golden age” of dungeon crawlers. Early game developers loved to develop these games because they were simple to code, required little if any graphics and were fun. Thus, every gamer has at least one dungeon crawler that they love, be it the modern *Diablo 3* or the old 1981 *Wizardry*).

To honor these games, the application is a classic style dungeon crawler. *Dungeon Crawler* is currently a simple game that has a set number of rooms, and has a set number of rooms. The objective is to navigate through the dungeon, whilst fight off monsters, to fight the boss.

1. **Gameplay:**

The player start the game by typing in anything and hitting enter. Then, the player is required to set up the skills of their character. The player has a total of 10 points that can be distributed amount health, attack, dodge, and luck. This distribution will determine how effective certain actions later in the game are.

After the skill set up, players may purchase items from the store. There are three kinds of items in the game. The first will boost player skill points. The second can be used in combat. The third is an item that will revive the player upon death.

The player is now ready to enter the dungeon. They are initially required to move one “O” up to enter the dungeon. However, from now on, the player decides where to move to. With every move, there is a chance that a Monster will spawn. Once a monster spawns, the play can decide to engage it, or try to sneak by. If sneaking fails or the player engaged the monster must be defeated to move one. The final room is the boss room. The player must defeat the boss in 8 turns. The player also collects points throughout the game for accomplishing certain tasks. Points are later shown on the score board if the player won the game. The game ends if the boss is defeated (win), the player ran out of boss fight turns (loose), the player dies at any point without the revive item active (loose).

1. **Code Content:**
2. **Number of variables and lines:**

|  |  |
| --- | --- |
| Line of Code (Just C++ code) | 631 |
| Line of Just comment (lines made of only comments) | 70 |
| Total Number of comment lines (including ones after C++ code) | 194 |
| Empty lines | 27 |
| Total lines of Code (Total lines in main) | 728 |

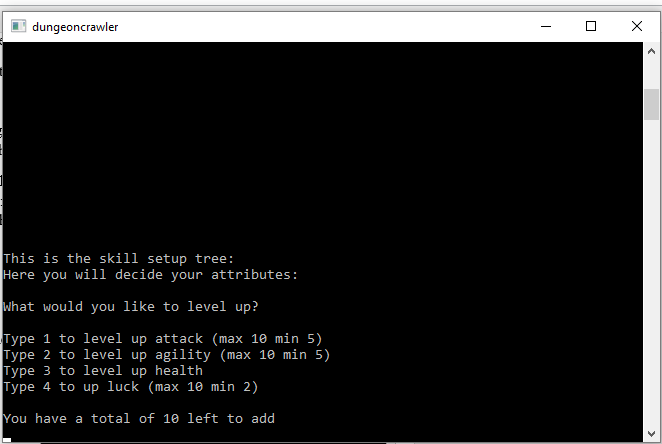
* Number of Variables: 59

1. **Table of concepts:**

*Disclaimer: The location will lead to one example of the concept being used. The concept may be used more frequently throughout the code.*

|  |  |
| --- | --- |
| Constructs | |
| Do while loop |  |
| While loop |  |
| For loop |  |
| if – else statements |  |
| Turnery Operator |  |
| Switch Case |  |
| If statements (in singular form) |  |
| Primitive Data Types | |
| Short |  |
| Int |  |
| Char |  |
| String |  |
| Float |  |
| Booleans |  |
| File-streaming | |
| ifstream |  |
| oftream |  |
| .open(“”) |  |
| .close() |  |
| .clear() |  |
| in |  |
| out |  |
| Math Function and Random Number Function | |
| sqrt( ) (<cmath>) |  |
| rand() |  |
| srand() | Line 27 |
| Time Function | |
| time() | Line 27 |
| Formatting | |
| setw() |  |
| setprecession() |  |
| fixed |  |

1. **Code Break Up:**



1. **Skill Set Up:**

As I previously mentioned, after the title screen the user sets up their skills. A deeper look, into the process will now be discussed.

The user will enter a number (1 to 4) that corresponds to the skill they would like to add points to. A switch case to check what skill was selected.

The range

Then, they decide how many points they will add to the selected skill. There is a do-while loop in place here, to ensure that the amount entered does not exceed the number of points available. The points are then added to the skill, and removed from the spendable points. Then one of three conditions could execute.

The first, the number of points in the skill fit into the range and no changes are done.

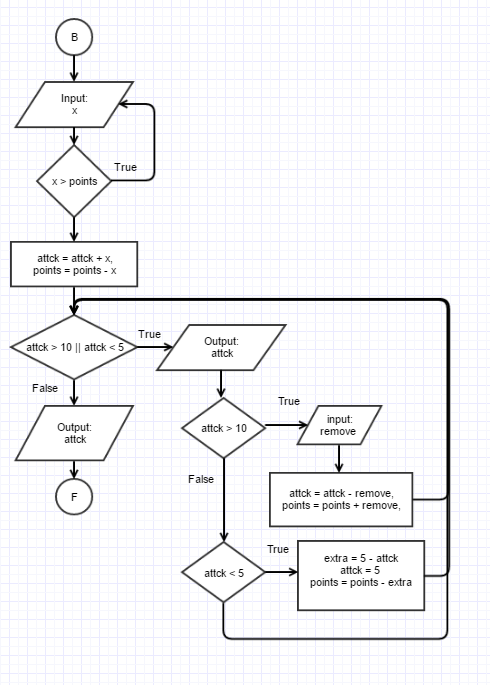
The second, there are too many points in the skill. The program will ask they user how many points they would like to remove. The removed point will then be added back to the spendable points.

The third, there are too few points in the skill. This will only happen when the user removes too many points. The skill will then just reset itself to its minimum amount. All the extra points in the spendable points will be cleared.

The program will continue to switch between condition 2 and 3 until the first condition is met. If else statements are used to decide which condition to execute. This will happen to all the skills except for the “Health” skill.

The “Health” skill has no range, so the number of points added by the user will be added to *health*.

As you can see, if else statements and do while loops are used extensively.



Pseudo code:

Flowchart if the player chooses to upgrade the Attack skill.

user inputs number of points to add;  
switch (input){  
case 1,2,4: (1, 2, 4 are individual cases)  
 Points are added to the selected skill.  
 Points are removed from spendable   
 points.  
 while points (are not in range){  
 if (skill points > range) {  
 Ask user to remove points   
 from the skill, and correct the  
 amount of spendable points.  
 }  
 else if (skill points < range){  
 Store the extra points then  
 reset the skill and remove the   
 extra points from the total   
 spendable points.

False

}

}

case 3:  
 Add the points health.

}