Text-Based Adventure Game

1 Introduction

You will have two projects this semester: a midterm project and a final project. The final project will build off the work that you do in the midterm portion of this project. You must submit the first part in the middle of the semester and the second part by the end of the semester. The final project will be handed out at a later date.

2 Teamwork

This project should be completed in a group of 4 students maximum. The workload must be divided amongst the group equally or else the group will have to be changed. In the report you will have to detail the work done by each student.

Each group should only submit one copy with a list of names of every member in the group report file name.

3 Design Tasks

Before the modern video games as we know today, there was the text-based adventure. The text adventure is the earliest form of interactive fiction to exist on a personal computer. The first one of these was created in 1975 by William Crowther called *Colossal Cave Adventure*. Games like *Zork* or *The Hitchhiker's Guide to the Galaxy* would follow. Your task for this project will be to develop a text-based adventure game in C++ using your knowledge of everything that we have discussed so far.

Project Objectives

- Practice analyzing and debugging techniques.
- Develop good coding habits:
  - Use separate code files for the declarations and implementations of different classes.
  - Use techniques to make the code concise and modular.
  - Use comments where necessary.
  - Use proper indentation.
- Read/write data from/to files.
- Basic interface design with user selections.
- Implement object-oriented programming concepts such as inheritance, overloading.
- Work with a team

4 Coding / Code Design

You will create a text-adventure game with some very simple and basic features. In the final project you will continue working on this project. You will add more features to make this project more complicated as well as change some features later using more advanced theory.

This game will have you create a player character to fight enemies and advance through 4 floors to reach the end. You can add whatever theme you want to the game you create, as long as you meet the minimum requirements mentioned. The overall structure of this project should be the same, but the experience for each game made can be made as creatively as you want. You may feel free to make any modifications you like, so long as the basic structure remains unchanged.

4.1 Classes:

Player Class

This class should contain information about the player.

You should create the following member variables:

- Player Name
- Player Maximum Health
- Player Current Health
- Player Attack Damage

The member functions should do the following:

- Setters for all the member values. This will allow you to change the member value through this member function.
- Getters for all the member values. This will allow you to view the value through this member function.
- A member function to add an inputted value to the player's current health
- A member function to add an inputted value to the player's attack damage
- Operator overload of the operator<< to print out all the member variables from the player class.

```
class player
{
    private:
      string Name;
        //This should hold the player's name
        int MaxHealth;
        //This should hold the maximum health of the player
        int HealthPoints;
        //This should hold the current health of the player
        int AttackDamage;
        //This should hold the attack damage of the player



    public:
        player();
        //default constructor

        string getName();
        int getMaxHP();
        //returns the maximum health
        int getHP();
        //returns the current health
        int getDMG();
```

```
        //returns the current damage

        void setName(string);
        void setMaxHP(int);
        void setHP(int);
        //sets the current health
        void setDMG(int);
        //sets the current damage

        void ModifyHealth(int);
        //Increments or decrements the player's health by the input value
        void ModifyDamage(int);
        //Increments or decrements the player's attack damage by the input value

        friend ostream& operator <<(ostream& os, const player& p);
};
```

Enemy Class

The enemy class should have all the features as the player class, except the enemy should also have a level member variable. This class should contain all the information about the enemy.

You should create the following member variables:

- Enemy Name
- Enemy Maximum Health
- Enemy Current Health
- Enemy Attack Damage
- Enemy Level. Level should correspond to the expected difficulty of the enemy.

The member functions should do the following:

- Setters for all the member values. This will allow you to change the member value through this member function.
- Getters for all the member values. This will allow you to view the value through this member function.
- A member function to add an inputted value to the enemy's current health
- A member function to add an inputted value to the enemy's attack damage
- Operator overload of the operator<< to print out all the member variables from the enemy class.

```
class enemy
{
       private:
        string Name;
        //This should hold the player's name
        int MaxHealth;
        //This should hold the maximum health of the player
        int HealthPoints;
        //This should hold the current health of the player
        int AttackDamage;
        //This should hold the attack damage of the player
```

```
        int level = 0;

      public:
            enemy();
        enemy(string, int, int, int, int);

        string getName();
        int getMaxHP();
        //returns the maximum health
        int getHP();
        //returns the current health
        int getDMG();
        //returns the current damage
        int getLVL();
        //returns the level

        void setName(string);
        void setMaxHP(int);
        void setHP(int);
        //sets the current health
        void setDMG(int);
        //sets the current damage
        void setLVL(int);
        //sets the level

        void ModifyHealth(int);
        //Increments or decrements the player's health by the input value
        void ModifyDamage(int);
        //Increments or decrements the player's attack damage by the input value

        friend ostream& operator <<(ostream& os, const enemy& e);
};
```

Weapon Class

This class is to hold information relating to a weapon that you should be able to equip and use later. You will have to make a 2D matrix using vectors using the Weapon class. You should create the following member variables:

- Weapon Name
- Weapon Damage

The member functions should do the following:

- Setter member functions
- Getter member functions

```
class weapon
{
    private:
        string W_item;
        //Name of the weapon item
        int W_damage;
        //Attack damage to the item. This can be added or removed by the player

    public:
        weapon();
        weapon(string i, int d);
```

```
        void setItem(string);
        void setDMG(int);

        string getItem();
        int getDMG();


};
```

Scoreboard Class

This class should hold information about the score the player has gained over the course of the game. This information should be updated whenever the player defeats an enemy or changes to a new floor. Remember, the scoreboard information needs to be stored onto a file.
, the, the difficulty, and floor.
You should create the following member variables:

- Player Name. Set at the beginning of the program.
- Score of the player. Score should increase after defeating enemies.
- Difficulty. Set at the beginning of the program. Overall difficulty of the game is stored here.
- Difficulty Modifier. Depending on the selected difficulty, this modifier should be a factor that multiplies the standard base health and damage of an enemy.
- Floor. Floor denotes the area or zone that the player is at. Higher floors imply more difficult enemies.

The member functions should do the following:

- Setter member functions
- Getter member functions
- A function to add points to the overall player score by passing in a value.
- A function that ONLY outputs: Player Name, Difficulty, and Floor. Not the Difficulty Modifier.
- Operator overload of the operator<< to print out ALL the member variables.

```
class scoreboard
{
	private:
		string SB_Name;
		//Name of the player
		int SB_Score;
		//score of the player. Score should increase after the player defeats an
enemy or through some custom method designed by you.
		//(*Note: Higher level enemies should give more points.)
		string SB_Difficulty;
		int SB_DifficultyMod;
		//Difficulty of the game.
		int SB_Floor;
		//keeps track of the area in the game the player is currently at


	public:
		scoreboard();
```

```
        //default constructor

        void setName(string);
        void setScore(int);
        void setDiff(string d);
        void setDiffMod(int);
        void setFloor(int);

        string getName();
        int getScore();
        string getDiff();
        int getDiffMod();
        int getFloor();

        void AddScore(int);

        void SB_out();

        friend ostream& operator <<(ostream& os, const scoreboard& p);
};
```

FileOperations Class

This class should handle everything relating to file manipulation. The opening and writing to files is done using this class.

You should create the following member variables:

- The filename of the scoreboard file
- An fstream variable for the scoreboard file
- The filename of the player file
- An fstream variable for the player file

The member functions should do the following:

- Functions to output the current progress to a file
- Functions to ask the user if they want to create a new file or open an existing file. This function should make checks to see if all the appropriate files exist to start.
- A function to close all the opened files.
- A function to search for all the scores of previous games and output them to the screen. This function should sort the scores in order. Vectors will be necessary for this part.

```
class FileOperations
{
	private:
		string SB_file_name = "scoreboard.txt";
		//holds the name of the scoreboard file
		fstream SB_myFile;
		//opens the scoreboard file

		string P_file_name;
		//holds the name of the player file
		fstream P_myFile;
		//opens the player file

	public:
```

```
        void Save2File(player&);
        //Function Saves player progress into a file
        void Save2File(scoreboard&);
        //Function Saves scoreboard progress into a file

        void ChooseFile(scoreboard&, player&);
        //Function that asks the user to select the file to use

        void CloseFile();
        //Function to close the files

        void ScoreRank();
        //Function to output scores from all the scoreboards
};
```

4.2 Main Functions:

Player Combat:

In this project you will have the player fight enemies. This function should continuously loop until the combat encounter ends. In this loop the player needs at least 4 options.

- The player must be able to examine themselves and view their maximum health, current health, and attack damage.
- The player must be able to examine the enemy and view their maximum health, current health, attack damage, and level.
- The player must have the option to attack the enemy. The attack should be followed by the enemy's turn
- The player must have the ability to run away from the encounter.

```
void PlayerCombat(player&, weapon[4][4], enemy&, scoreboard&);
```

Enemy Combat:

In this project you will have enemies fight the player. When this function is called the enemy should attack the player. The player's current health should be reduced.

This function should examine the player's current health to see if it drops to zero or lower, then end the function can save and end the program displaying the final score. If the enemy's current health does not fall to zero or below, then the function returns true. If the enemy's current health does fall to zero or below, then the function returns false. This function should work with the PlayerCombat function.

```
bool EnemyCombat(enemy&, player&, scoreboard&);
```

Player Options:

When the player is not in combat the player should have access to an option menu. There should be 4 options in this menu.

- Examine self. The player can view their own health and damage.
- Continue. The player should be able to continue playing the game.
- Scoreboard. The player can see the scoreboard: score, difficulty, and floor
- Quit. All the progress will be saved onto files and then the program will end
-

```
void PlayerOptions(player&, FileOperations&, scoreboard&);
```

Choose Weapon:

In main you should create a 2D array using the Weapon class. This function should allow the player to pick a weapon from the 2D array and set that as the attack damage for the player. Depending on the Floor, the selection of weapons should change.

```
void ChoooseWeapon(weapon[4][4], int, player&);
```

4.3 Main:

The main functions should be called multiple times throughout the main of the code. In this game there should be at least one enemy per floor and 4 floors total.

4.3.1 Give the user the option to create a new player or access a previously saved file of a player.

4.3.2 When starting a new game, the user should be able to select the difficulty level of the game. Difficulty level is held in the Scoreboard class. You should create a variable that will modify all the attack damage of the enemies depending on the difficulty level selected.

4.3.3 Create the floors and enemies on those floors for the player to fight. (Floors is just a term to describe the area or zone that a player is in.)

4.3.4 The player should be able to change their attack based on a weapons list of some sort. Create a 2D vector using weapon class. This should be a 2D array matrix with different weapons storing the Damage points associated with each weapon. The weapons available should change based on the floor, so weapons with lower attack damage should only be available on lower floors. The higher the floor the higher the weapon attack available.
For example:
{ {Fist, 10}, {Rock,15}, {Knife, 20}, {Spear, 25},
{Sword, 30}, {Axe, 40}, {Gun, 50}, {Bomb, 80} }

4.3.5 The player options function should be called multiple times. These options should be before an enemy encounter.

Additionally, there should be member functions from the Fileoperations class being called to save the progress of the game by inputting the current player member variables and the current scoreboard variables to files.

4.3.6 At the end of the game the score of the player should be sorted into a scoreboard. The scoreboard should then be displayed for the user to see. This should be done using a member function from the FileOperations class.

***Note:** Make sure that the game that you create is well tested and can be beaten (Meaning: You can reach the end of floor 4).

4.4 Bonus:

- Create a class that adds random events before a combat encounter
- Create a class and member function within it with events to select from
- Use the rand() function
- The random function should give a new event each time the member function is called
- An event should ask for player input and have the results effected by the input
- These random events should occur at least once per floor

You will be awarded with 5 additional bonus points.

5 Code

TAs will run your code and then test for any errors. Your project code should have the following qualities:

• All codes should be completed by yourselves. TAs will read your codes, plagiarizing work from other groups or other resources is forbidden.

• Make sure your codes can be compiled successfully before you submit them.

• Your codes should be concise and modular. All files, including .cpp, .h and all data files, should be submitted in a project folder, and all codes in one main file is not recommended.

• Provide comments in your codes where necessary. This will make understanding your code easier for TAs when grading.

• Include a readme file to explain how everything works.

You may send a zip file of your entire project, but that is not necessary. What is necessary is that the project must have all the .cpp files and .h files, and any other additional data files needed to compile your code.

Sending the entire project may be the easiest way to check your code as well as the easiest way to avoid any missing files.

6 Report

TAs will read your report and then check your codes accordingly. Your project report should describe clearly:

• Architecture of your project and the explanation of each section it has. Please explain how some of the functions/classes work. Explain the designing logic behind the codes.

• Specification of classes, including the attributes and functions of each class.

• System functions you implement, including how parameters are passed when each function is called.

• All the highlights of your projects. Some of the students may have some features that are not exactly the same as the requirements from the project and they think they are good to be included. That is fine, but you need to explain well.

• Important: always explain why you code like that.

• Detail of the work each student has done. You should explain which section of code you were responsible for.

• Notice the system you are using (Windows, Mac, MS, or xcode).

7 Grading

The grading scale of the first submission is:

- Code (70 points)
- Report (30 points)
- Bonus (5 points)

	Excellent	Average	Poor	Points
Code	All of the requested functions are implemented and work as they should. No elements are missing. The code is compiled with no errors.	Most of the requested functions are implemented and work as they should. Occasionally there are errors that were not fixed in debugging.	There are missing elements. Only a few functions are implemented. There are errors throughout the code.	70
Report	All work is documented and explained.	Some of the work is documented. There are missing explanations of the work.	Very little work is documented. Explanations of what was done are missing.	30
				Total: 100