Scott Upman

03/20/2021

Project Title: RPG Create a Character/ Auto Boss Generator

Summary of Project: The project is inspired by Massive Multiplayer Online Role Playing Games (MMORPG) to create a RPG character that traverses through boss fights. As the user launches the program, the user can log in or register if an account hasn't been made. Then a character can either be selected to play or a new character can be created. There will be multiple choices for the type of character that the user wants to create, using inheritance and polymorphism to modify the members within the superclass and subclasses.

There will be a text file that stores the username, password, and information of the characters that were created. The player's level from the created character will be used to generate the bosses within the game, associating the player's level with the boss stats. You could think of this as an endless boss generator that the player levels up after defeating each boss and updates the boss stats to make the game scale with the level of the player.

To preserve the originality of the project I will mostly be referencing the textbook and the C++ documentation since I want to implement a project like this in my own way. If I were to reference any websites it would be to understand how something in my program could be improved rather than blatantly copying code from a website. I also will create small test programs within Visual Studio to understand how to write something that I want to implement into my game. In informal terms I like to call it "sandboxing".

Classes

1. **Character –** The class will serve as superclass for the different types of characters that will derive from this class. This class will serve as the players base stats.
    a. **Warrior**
    b. **Mage**
    c. **Archer**

    Warrior, Mage, and Archer classes will utilize inheritance and polymorphism to allow the character to contain different initialized stats based off which class was instantiated.

2. **Account –** The class will handle Input from the user with the username and password and will also hold the characters associated with that account.

3. **Boss –** The boss class will have stats that scale with the player level from the Character class.

4. **GameManager –** This class will handle the background operations with the program such as leveling up the player, regenerating skill points each round, determining when the game is over and so forth.

5. **FileManager –** The FileManager class will manage I/O operations within the text file and will continuously update the file so that the player can save their progress and continue where they left off.

6. **Helper –** The Helper class will validate user input and perform operations such as selecting the character, creating a new character, prompting for username and password and such.

7. **StatsTemp –** The StatsTemp class will hold temporary values of the player's current stats. This will be used for reloading and holding the player's stats for when they level up or when they want to relaunch the game.

Character

Abstract class: No

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
HP	**Int**	**No**	**Hold's the player's health points**
ATK	**Int**	**No**	**Hold's player's attack points**
DEF	**Int**	**No**	**Hold's player's defense points**
name	**String**	**No**	**Contains character's first and last name**
SP	**Int**	**No**	**Will hold the player's skill points for choosing which attack to use**
Level	**Int**	**No**	**Hold's the player's level that will be used for generating the bosses**
Role	**String**	**No**	**Will hold the type of character that player created**
MP	**Int**	**No**	**Will hold the player's mana points**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
int get_HP()	No	No	No	No	Returns the health points stored in HP
Int get_ATK()	No	No	No	No	Returns the attack points stored in ATK
Int get_DEF()	no	No	No	No	Returns the DEF points stored in DEF
string get_name()	No	No	No	No	Read only function returns the first and last name of the character

Int get_SP()	No	No	No	No	Returns the skill points stored in SP
Int get_level()	No	No	No	No	Returns the level associated with the character
string get_role()	No	No	No	no	Read only function Returns the string stored in role
void set_HP(int HP)	No	No	No	No	Sets the HP variable to the integer parameter
void set_ATK(int ATK)	No	No	No	No	Sets the ATK variable to the integer parameter
void set_DEF(int DEF)	No	No	No	No	Sets the DEF variable to the integer parameter
void set_SP(int SP)	No	No	No	No	Sets the SP variable to the integer parameter
Void initialize_base_stats()	No	Yes	No	No	Initializes the character's stats based on the different type of character that was instantiated.
Void display_stats(Character& character)	No	No	No	No	Displays the character's stats while being defined outside of the class.
Void display_attacks(Character& character)	no	Yes	No	No	Will display the names of the attacks
Void Attack(Boss& boss)	No	Yes	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Void secondary_attack(Boss& boss)	No	Yes	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.
Void final_attack(Boss& boss)	No	Yes	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Warrior

Abstract class: No

Subclass of: Character

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
Basic_attack	**String**	**No**	**Will store the name of the basic attack. These variables will be pre-defined.**
Special_attack	**String**	**No**	**Will store the name of the special attack**
Ultimate_attack	**String**	**No**	**Will store the name of the ultimate attack**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`Warrior(string name, string role, int level, int SP, int ATK, int DEF, int HP, int MP);`	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Warrior(string name)	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Void initialize_base_stats() override	No	No	No	No	Function will initiate the character stats based off which subclass was instantiated
Void attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Void secondary_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.
Void final_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Mage

Abstract class: No

Subclass of: Character

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
Basic_attack	**String**	**No**	**Will store the name of the basic attack. These variables will be pre-defined.**
Special_attack	**String**	**No**	**Will store the name of the special attack**
Ultimate_attack	**String**	**No**	**Will store the name of the ultimate attack**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`Mage(string name, string role, int level, int SP, int ATK, int DEF, int HP, int MP);`	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Mage(string name)	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Void initialize_base_stats() override	No	No	No	No	Function will initiate the character stats based off which subclass was instantiated
Void attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Void secondary_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.
Void final_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Archer

Abstract class: No

Subclass of: Character

Composed of: N/A

Variable Name	Data Type	Static	Description
Basic_attack	**String**	**No**	**Will store the name of the basic attack. These variables will be pre-defined.**
Special_attack	**String**	**No**	**Will store the name of the special attack**
Ultimate_attack	**String**	**No**	**Will store the name of the ultimate attack**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`Archer(string name, string role, int level, int SP, int ATK, int DEF, int HP, int MP);`	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Archer(string name)	No	No	No	No	Constructor for the Warrior class that will initialize the character's stats
Void initialize_base_stats() override	No	No	No	No	Function will initiate the character stats based off which subclass was instantiated
Void attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.
Void secondary_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then

					modifies the boss's health.
Void final_attack(Boss& boss)	No	No	No	No	Generates a number to deal damage to the boss, then modifies the boss's health.

Account

Abstract class: no

Subclass of: N/A

Composed of: Character

Data Members

Variable Name	Data Type	Static	Description
username	**String**	**No**	**Will hold the user input for the username**
password	**String**	**No**	**Will hold the user input for the password**
Vector<Character*> characters	**Character**	**No**	**Will hold the character pointers within the vector**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
~Account()	No	No	No	No	Destructor for Account class that deallocates the pointer vector
Account()	No	No	No	No	Constructor for the Account class
Account(string username, string password)	No	No	No	No	Constructor for the Account class
Void Operator+(const Character& right)	No	No	Yes	No	will use push_back() function for inserting the Character object into the vector
`ostream& operator<< (ostream& out, Account& account);`	No	No	Yes	No	Overload operator that writes to account to the output file
`istream& operator>> (istream& in, Account& account);`	No	No	Yes	No	Overload operator that reads data from input file into account

Vector<Character*> get_characters()	No	No	No	No	Returns the vector that stores all the character pointers.
string get_username()	No	No	No	No	Returns the username
string get_password()	No	No	No	No	Returns the password

Boss

Abstract class: No

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	**Data Type**	**Static**	**Description**
HP	**Int**	**No**	**Will store the health points within HP variable**
ATK	**Int**	**No**	**Will store the attack points within ATK variable**
Level	**Int**	**No**	**Stores the level of boss**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Boss(Character* character)	No	No	No	No	Constructor for the boss class based off Character object
Int get_HP	No	No	No	No	Returns the health points stored in HP
Int get_ATK	No	No	No	No	Returns the attack points stored in ATK
void set_HP(int HP)	No	No	No	No	Sets the HP variable to the int parameter
Void attack(Character& character)	No	No	No	No	Generates a number to deal damage to the Character

GameManager

Abstract class: No

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	**Data Type**	**Static**	**Description**
Game_over	**Bool**	**Yes**	**Variable will hold a bool value that determines if the player loses**

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Void level_up(Character& character)	Yes	No	No	No	Function will increment player level by 1, and will prompt user for which stats to upgrade
Void Update_stats(Character& character)	Yes	No	No	No	Stats of the boss will be updated and the player will have the option to assign skill points to the data members within Character class
void set_game_over()	Yes	No	No	No	Setter for the game over static variable
Bool get_game_over()	Yes	No	No	No	Program will end when game_over variable is set to true

FileManager

Abstract class:

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`bool search_account(ifstream& infile, string username, string password)`	yes	No	No	no	Function will search through the text file for the username and password
`bool search_username(ifstream& infile, string username)`	yes	No	No	no	Function will search through the text file for the username
`void set_file_position(ifstream& infile, string username)`	Yes	No	No	No	Function will search for the username and set the stream position back to the beginning of the read line

StatsTemp

Abstract class:

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
HP_temp	**Int**	**Yes**	**Stores the temporary value of the character's HP**
ATK_temp	**Int**	**Yes**	**Stores the temporary value of the character's ATK**
DEF_temp	**Int**	**Yes**	**Stores the temporary value of the character's DEF**
MP_temp	**Int**	**Yes**	**Stores the temporary value of the character's MP**
SP_temp	**int**	**Yes**	**Stores the temporary value of the character's SP**

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`void store_stats(Character& character);`	yes	No	No	no	Stores the stats from the character into the temp variables
`Void load_stats(Character& character);`	yes	No	No	no	Loads the values stored in temp variables to the data members for the Character
`int get_SP() { return SP_temp; }`	Yes	No	No	No	Returns value stored in SP_temp

Helper

Abstract class:

Subclass of: N/A

Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
`void validate_attack_selection(int& selection);`	yes	no	no	no	Validates user input for attack selection
`Void validate_role_selection(int& selection);`	yes	no	no	no	Validates user input for role selection
`create_character(Character*& character, Account* account);`	yes	no	no	no	Creates a character and stores that character into the account
`prompt_user(string& username, string& password);`	yes	no	no	no	Prompts for username and password
`Character* select_character(Account* account);`	yes	no	no	no	Gets the character stored in the account
`void validate_menu_select(int& selection);`	yes	no	no	no	Validates menu selection
`void validate_character_prompt(int& choice);`	yes	no	no	no	Validates character selection

Demonstration of OOP Concepts

1. **Encapsulation**
   The program will have protected data members in the Character class to allow the derived classes to have access to those data members. Any modification applied to these stats outside of the derived classes will use setters and getters since they are public member functions within the Character class. This encapsulation will ensure that no data members are corrupted. For the other classes, any private members that need to be modified will include setters and any private members that need to return data will use getters.

   - Character.h, line 12-33
   - Boss.h

2. **Inheritance**
   Inheritance will be used to transfer public and protected members to the derived classes to form "is a" relationships with the superclass. The subclasses will affect the data members of the Character class by performing operations with the setters and getters they inherit (Archer, Warrior, Mage class).

   - Archer.h/.cpp
   - Warrrior.h/.cpp
   - Mage.h/.cpp

3. **Polymorphism**
   Polymorphism will be used to initialize the data members in Character based off which subclass is instantiated when the player picks which "role" that they want to play as (Warrior, Mage, Archer). This will provide flexibility to the functions that take a Character& parameter since Warrior, Mage, and Archer have an "is a" relationship with Character.

   - Character.h, line 15
   - Mage.h, line 13, 21-23
   - Warior.h, line 14, 24-26
   - Archer.h, line 14, 23-25

4. **Static Members/Functions**
   Classes that have no need to be instantiated, such as the GameManager and the FileManager, will be treated as static and will handle the game's background operations such as displaying the user interface, updating the player's stats and so forth. Functions that belong to the class will be explicitly called with the class scope. This will show clarity on where the function is being called from.

   - FileManager.h/.cpp

- GameManager.h/.cpp
- StatsTemp.h/.cpp
- Helper.h/.cpp

5. **Friend functions**
   Friend functions are used with overloaded operators to be able to overload the << and >> operators. This allows for the contents of the account class to be read from a file or written to a file using an operator instead of having to call a function to perform the read/write operation. The account object is passed as an argument and allows access to the private members of the classes where the friend function was declared

   - Account.h – line 30-31
   - Account.cpp – line 22 - 72

6. **Overloaded Operators**
   Overloaded operators in this program will be used to store Character objects into a private STL container within the Account class. They are also used to write/read the contents of the account class to/from a .txt file.

   - Account.h, line 23, 30-31
   - Account.cpp, line 22-72

7. **Text Files**
   The text files are used to initialize data to the Account class's data members. An input file will be opened at the beginning of the program and there will be a temporary output file that will eventually replace the input file at the end of the program. After the user is done using the program, all the data stored in the character object will be stored in the vector within the Account class and the characters within the vector will have their data written to a text file. From that point when the user wants to launch the program again, data will be read from the text file and stored into the account class's data members and the process is repeated.