Kasen Ni

# Report

The project consists of six different classes, including Player class, weapon class, enemy class, file operation class, and scoreboard class, and the main file, which is where the main function is located. Each class stores and controls the values of the specific elements that are essential to the whole game.

**Player class:**

Private:

The player class stores and controls the values of 5 basic elements of the player. These elements are player name, Max health, current health, attack damage, and currency. Among these elements, the max health is initialized to 200 and will not be changed throughout the game.

Public:

In the default constructor, the player is initialized to have the name “New Player” with 200 max health, 200 current health, 0 attack damage, and 0 currency.

There are 5 setter functions, 5 getter functions, 3 modifier functions, and 1 print function in the public.

The 5 setter functions are used to set the values of player name, max health, current health, attack damage, and the currency by taking one parameter that corresponds to the certain element type.

Each of the 5 getter functions returns a certain element defined in the Private and takes no parameter.

The 3 modifier functions increase the values of current health, player’s attack damage, and currency amount. They all take one parameter and increase the value of the corresponding element by adding the parameter to the current value of the element.

The print function prints out the current values of player name, max health, current health, attack damage, and currency into the screen.

The operator << function stores the player information into the player file.

**Weapon Class:**

Private:

The weapon class contains 4 private variables. They are weapon names, weapon damage, the weapon uses, and weapon cost.

Public:

The default constructor initializes the weapon's name to be "Fist", which has 10 damage, one use, and 0 costs.

The other default constructor has two parameters. They are weapon-name in type string and weapon damage in type integer. When the function is called, the value of the weapon name parameter will be assigned to the weapon name and the value of weapon damage will be assigned to the weapon damage.

There are 4 setter functions that set the values of weapon name, damage, uses, and cost by taking one parameter into the function. When the setter function is called, the certain variable of the private part will be set to the value of the parameter argument.

There are also 4 getter functions that return the values of weapon name, weapon damage, uses, and costs. They do not have a parameter.

**Enemy Class:**

The enemy class has a base class called player class.

Private:

The enemy class has 6 private variables. They are enemy name, enemy max health, enemy current health, enemy damage, enemy level, and bounty. Enemy name is the name of the enemy. Enemy max health is the max health of the enemy. This variable is different from the player's max health and will be changed throughout the entire game depending on the current level of the enemy. Enemy damage will change depending on the current level of the enemy. When the enemy is defeated, the bounty of the enemy will be added directly to the player's currency.

Public:

The default constructor of the enemy initializes the enemy to have a name of "Goblin 1", with 5 max health, 5 current health, 2 attack damage, level 1, and 5 bounties.

There are 6 setter functions for the enemy classes. Each of them sets the value for one of the 6 private variables. They take one parameter and set the certain private variable value to the value of the parameter.

There are 6 getter functions. Each of them returns one of the six private variables.

There are also two modifier functions that modify the value of current health and attack damage of the enemy. The two modifier functions take one integer parameter and set the current health or attack damage to the value in the parameter.

void printEnemy(): this function does not take any parameter and does not return anything. It only prints out the enemy's information (the six private variables) into the screen.

friend void operator << (ostream&, const enemy&): this function is a friend function in the enemy class. It stores enemy information into a file directly.

void operator << (stack<enemy>&): this function takes a stack of enemy type as a parameter. It simply prints out everything inside the enemy stack and clears the stack.

**ScoreBoard Class:**

Private:

The scoreBoard class has 5 private variables. They are:

- Scoreboard Name: represents the player's name on the scoreboard.
- Scoreboard Score: represents the score that the player has earned for the game.
- Scoreboard Difficulty: the difficulty mode of the game (easy, medium, difficult).
- Difficulty Modifier: a multiple that will decide how the enemy's max health, attack damage, and bounty will increase based on the difficulty mode the player picked at the beginning of the game.
- Scoreboard Floor: the current floor the player locates. As the player defeats an enemy, he or she will move to the next floor. Once the enemy on floor 4 is defeated, the game is over and the player wins.

Public:

The default constructor initializes Scoreboard Name to be "New Player", Scoreboard score to be 0, Scoreboard difficulty to be "easy", Scoreboard difficulty modifier to be 1, and Scoreboard floor to be 1.

The five setter functions set the 5 private variable values by taking one parameter and assigning it to the corresponding private variable.

Each of the five getter functions returns the value of the corresponding private variable. They don't have any parameters.

The addScore function takes one integer parameter. It adds the scoreboard score by the integer value of the parameter. The function does not return anything.

The SB_out function returns nothing. It simply prints out the information of the 5 private variables into the screen.

bool operator < (const scoreBoard& s1): This function returns the comparison between the scores of two scoreboard objects. The function is used to help sort the list of type scoreboard based on the scores of the players from highest to lowest.

**FileOperation Class**

Private:

SB_file_name: a private variable in string type and is defaulted to have a value of “scoreboard.txt”. This value will not be changed throughout the entire game.

SBFile: a private variable in type fstream. The variable holds and opens the “scoreboard.txt” file.

P_file_name: a private variable in type string. It holds the name of the player file. The value of P_file_name is read from user input.

PFile: a private variable in type fstream. The variable holds and opens the player file, while will store the information, such as player name, max health, current health, attack damage, currency, and weapon, of the player.

Public:

Save2File: this function doesn’t return anything. It takes two parameters. One of them is in type Player, which is defined by the Player class. The parameter is a vector of type weapon. Type weapon is defined by weapon class. When opening the file, the old content in the file will be clear. The function stores the new information of the

player (player name, max health, current health, attack damage, currency), and new information of weapon in the weapon vector to the player file. When storing the weapon information, a “ * ” is placed at the end of the last weapon in the file so that the weapon information can be taken out from the file more conveniently when reading the information from the player file later on.

Save2File: this function has the same name as the previous function but the difference is that it only takes one parameter in type scoreBoard, which is defined by the scoreBoard class. The function stores the information of the scoreboard (scoreboard name, scoreboard score, difficulty mod, difficulty level) into the player file. Because the function is called after the previous Save2File function. Thus, there is no need to clear the file content when opening it.

ChooseFile: this function takes 4 parameters. They are s in scoreBoard type, p in Player type, list in a vector of type weapon, and shop in a vector of type weapon. When the function is called, it will first ask the user if he or she has an existing player file.

If the user answers “ Yes “, then the function will ask the user to enter the file name. If the file does not exist, then the function will terminate the program. If it does exist, then the function will import the information of the player and scoreboard from the player file to the corresponding variables. In the file, there are some null words like “Play name: ”, “ Max Health:”, and “Score:“. These words are just labels for the specific data and should be ignored when reading data from file to program. So a local variable called donCare is declared. All of the null words will be imported into donCare while other essential data will be imported to the specific variable. When reading data about the weapon, since the number of weapons that the player purchased is unknown each time when the player plays the game, thus a “ * ” is placed after the last weapon when storing the weapons into the file

and a for loop is used to draw out weapon data from the player file. This “ * ” works like a mark that tells the program when to stop reading the weapon data from the play file.

If the user answers “ No ”, then the function will ask the user to enter a name. This name will be used as the name for the new player file. Also, the name will also be the name of the player in the game. The function will automatically add “.txt” at the end of the name and create a new file for the player. All other basic elements of the player ( max health, attack damage, etc.) will be defaulted by the program.

CloseFile: this function simply closes the player file and the scoreboard file. It does not take any parameters or return something.

ScoreRank: this function reads data about players and their scores from the scoreboard.txt file and prints them out into the screen. Again, a local variable called donCare is applied to ignore the null words (such as “Player Name:”, and “ Score:” ) when reading data from a file.

- **Shop and ShopNode class:**

Basically, the Shop and ShopNode classes are the classes that create a doubly linked list for the shop feature in the game. The two classes are defined in the same file called shop.h and shop.cpp.

ShopNode class:

Private:

weapon Weapon: this private element stores the weapon

ShopNode* prev: this private stores the information of the previous shopNode in the linked list.

ShopNode* next: the private stores the information of the next shopNode in the linked list.

Public:

ShopNode(): this is the default constructor that sets prev and next to NULL.

ShopNode(weapon); this constructor takes one parameter of type weapon and sets the Weapon of the ShopNode class to be the weapon, the parameter.

void setNext(ShopNode* nxt); this function sets the next pointer of the ShopNode.

void setPrev(ShopNode* prv): this member function sets the previous pointer of the ShopNode.

ShopNode* getNext(); this function returns the next pointer.

ShopNode* getPrev(); this function returns the previous pointer.

string getWN(): this member function returns the weapon name of the ShopNode.

weapon getWeapon(): This function returns the weapon object information of the ShopNode.

void print(): This member function prints out the name, damage, and costs of the weapon in the ShopNode object.

Shop class:

Private:

typedef: define ShopNode* to be ShopNodePtr.

ShopNodePtr Head: this private is the head of the linked list.

ShopNodePtr Tail: this private variable is the tail of the linked list.

Public:

Shop(); the default constructor that sets head and tail to NULL.

void addWeapon(ShopNode*); add a ShopNode to the linked list.

void removeWeapon(weapon): remove a shopNode from the linked List based on the weapon input. The function will search for the shopNode in the linked list with the weapon data that is the same as the input weapon and remove it from the list.

void printAll(): print all the weapon information in the linked list.

void clearShop(): clear the linked list and shop.

weapon searchWeapon(string): search for the weapon in the linked list based on the input weapon name. If the weapon exits, then it returns the weapon object. Otherwise, it prints out that no such weapon exists in the list.

**Main File:**

Global variables:

In main File, 5 additional functions are declared and defined:

- EnemyCombat:

The function takes 10 parameters. They are enemy& e, Player& p, scoreBoard& s, vector <weapon> &w, list<scoreBoard>& SBList, stack<weapon> &tracker, Shop& shop, stack<enemy> &defeated, queue<string>& sideQuest, stack<int>& combatLog. When the function is called, it will call the midofyHP function in Player class, and use -1* enemy's attack damage as the argument to reduce the player's current health. In this way, the enemy finishes his attack. After that, the function will check if the player's current health is below or equal to 0. If it is below or equal to 0, then the function will save the player and scoreboard's information into the player file and scoreboard.txt file. Then it will sort the scoreboard.txt by the score from highest to the lowest. Finally, it will print out the enemy stack to show the enemies that the player has defeated in the game. It will also print out the string stack to notify the player of the side quest that he or she has completed in the game. After that, all the information in the stack, queue, or shop will be clear.

- PlayCombat:

There are 10 parameters for PlayerCombat. They are Player& p, enemy &e, scoreBoard& s, vector <weapon> &weaponList, stack<enemy> &defeated, list<scoreBoard>& SBList, stack<weapon>& tracker, Shop& shop, queue<string>& sideQuest, stack<int>& combatLog. At the beginning, the function declares a variable called enemyFlag at the beginning and sets it to false. This flag checks if the enemy has been defeated. The function uses a while (1) loop. Each time the loop is called, it will give the player 4 options: Attack, examine self, and examine enemy.

If the player enters ‘A’, the function will call the ChooseWeapon function so that the player can choose a weapon before fighting the enemy. Then the player will attack the enemy by calling the modifyHP function in enemy class and using the player’s attack damage time -1 as the argument for the parameter. After that, the function will check if the enemy’s current health is below or equal to 0. If it is, then it will call the modifyCurreny function in the Player class and the getBounty function in the enemy Class to increase the player’s currency. The enemy that is defeated will also be stored into an enemy stack that stores the enemies the player defeated. The stack will be printed out and deleted before the player leaves the game. Then the function will also call the addScore function in Scoreboard class to increase the score by 10 times * DifficultyModifier. Difficulty Modifier will change the basic elements (ex. Attack damage, max health) of the enemy depending on the difficulty mode the player picked at the beginning of the game. The function will record the damage the player has caused to the enemies, the currency the player has earned, and the number of rounds the player takes to defeat an enemy through the global variables cumulative_damage, cumulative_currency, and round_count. Next, the function will increase the floor by one and set enemyFlag to be true. The function will also check if the current weapon that is being used is Fist. If it is not Fist, then it will remove the weapon from the player’s weapon list and switch the current weapon

to Fist. Lastly, if the enemy's current health is greater than 0, then it calls the EnemyCombat function.

If the player enters 'S', the function calls the print function in the player Class to print out the player's information.

If the player enters 'E', the function will call the print function in the enemy Class to print out the enemy's information.

If the player enters 'C', the function will call another function called chooseWeapon to allow the player to change the weapon.

At the end of the function, the function will check if the enemyFlag is true. If it is true, then the enemy of the current floor is defeated and it will end the while (1) loop.

- PlayerOptions:

The PlayerOptions function returns int result. It has 6 parameters. They are Player& p, scoreBoard& s, vector <weapon> &weaponList, Shop& shop, stack<weapon> &tracker, fstream& SBF. In the beginning, the function will give the player 5 options: Examine self, Continue, Buy from the shop, Scoreboard check, and Quit.

If the player chooses to enter 'E', the function will call the print function in the Player class and print out the information of the player. Then it calls the PlayerOptions function (itself) again.

If the player enters 'C', then the switch breaks and quits the function.

If the player enters 'B', then the function prints out the linked list that stores the weapon information. This list includes a list of weapons that the player can purchase if he or she has enough currency. Upon visiting the shop, the function will ask if the player wants to give up purchasing and quit the shop. If the player enters 'G', then the function calls itself again. If the player enters 'B', the function will ask the player to enter the name of the weapon that he or she wants to buy. After entering the weapon name, the function will search for the weapon and check if the player's currency is greater than the cost of the

weapon. If it is, then the purchase is successful, and the new weapon will be pushed back to the player's weapon list. If it is not, then the function will tell the player that the currency is not enough and call itself again.

If the player enters 'S', then the function will print out the information on the scoreboard.

If the player enters 'R', then the function will call the mainMenu function and take the player to the main menu.

- ChooseWeapon:

The function has two parameters. They are w1 for a vector of type weapon, and p in type player. When the function is called, a list of weapons from the vector of type weapon is printed out. These are the weapons that the player has purchased and has not been used up. Then the function will ask the player to choose a weapon by entering the name of the weapon. When the weapon name is entered, the function will search for the weapon with the same name. If it exists, then the attack damage of the player will be set to be attack damage of the weapon. If the weapon cannot be found or the player enters "Fist", the current weapon will be switched to Fist automatically by the default constructor.

- SBToList ( fstream& f, list<scoreBoard>& l) :

This function reads the players' names and scores from the scoreboard.txt file and converts these into scoreBoard objects. Lastly, the scoreBoard objects that contain the player's name and score will be stored into a list in type scoreBoard

- searchList(list<scoreBoard>& l):

This function will ask the player for a name. Then the function will search through the scoreBoard list for the player with that name. Then it will print out all the players with that name and their scores.

- operator << (list<scoreBoard> l, int i):
  This function prints out all the player names and scores in the scoreboard list.

- insertionSort ( fstream& f, scoreBoard s, list<scoreBoard>& l):
  This function uses insertion sort method to insert the score information of the player into the corresponding position of the scoreboard.txt file so that the scoreboard.txt file contains a list of player information arranged by the order of their scores from the highest to the lowest.

- averagedSB(list<scoreBoard>& oldList, list<scoreBoard>& averagedList):
  This function will read through the OldList to check if there is one or more players with duplicate names. If there is, then the function will take an average of these scores and store that information into a new list called averagedList. Everytime a player and the duplicates (if possible)  are checked, the player and the duplicates will be removed from the OldList to avoid rechecking the same player.

- addToStack(weapon removed, stack<weapon>& tracker):
  This function adds the removed weapon into a stack.

- printStack(stack<weapon> tracker):
  This function prints out the weapon name in the stack but not clearing the stack. It makes a copy of the tracker and the copy to perform the pop instructions.

- clearStack(stack<weapon>& tracker):
  This function clears the stack in type weapon.

- addToEnemyStack(enemy e, stack<enemy>& defeated) :
  This function adds an enemy object into the defeated enemy stack.

- clearEnemyStack(stack<enemy>& defeated):
  This function clears the enemy stack.

- mainMenu ( fstream& file, list<scoreBoard>& SBList, list<scoreBoard>& averagedList, stack<weapon>& tracker, stack<enemy>& defeated, Shop& shop, enemy& e1, Player& player, scoreBoard& scoreboard, queue<string>& sideQuest ):

  This function gives the user several options:
  (a) ScoreBoard sorted by score;
  (b) ScoreBoard sorted by name;
  (c) Averaged ScoreBoard;
  (d) Search by name;

(e) Enter game;
(q) Quit;
Each time the player chooses an option, the corresponding function will be called to get the expected result. However, when the player calls instructions a through d, the function will call itself again so that the player can continue to make options. When calling (d), the program will be terminated. But before the program is terminated, the function will save the player and scoreboard's information into the player file and scoreboard.txt file. Then it will sort the scoreboard.txt by the score from highest to the lowest. Finally, it will print out the enemy stack to show the enemies that the player has defeated in the game. It will also print out the string stack to notify the player of the side quest that he or she has completed in the game. After that, all the information in the stack, queue, or shop will be clear. Upon finishing performing all of the above tasks, the program will terminate.

- QuitGame (list<scoreBoard>& SBList, stack<weapon>& tracker, stack<enemy>& defeated, Shop& shop, enemy&e):
  This function is called at the end of the program, when it is called, it prints out the defeated enemy stack, sorted scoreboard list. Then it clears the defeated enemy stack, shop, and scoreboard list, and closes the files that are used during the game. Finally, it terminates the program.

- printSideQuest (queue<string>& sideQuest):
  This function prints out the side quest queue, which stores the information about the completed side quests. It also clears the side quest queue after printing it out.

- printCombatLog (stack<int>& combatLog, enemy& e):
  This function prints out the combat log information in the combatLog stack. The stack will be cleared after printing the combat log out.

- Main Function:

In the main function, 8 different types of weapons are defined at the beginning. Then these 8 types of weapons are put into a linked list class called the shop. Another vector of the weapon class called weaponList is created. This weapon vector is the player's list of purchased weapons. The player's weapon is default to be fist. This will not be shown in the weapon list.

When the player starts the game, the ChooseFile function in the fileOperation class will be called first to import data from the player file or create an empty play file. If there is data imported from the player file, the program will keep the player's currency and score information but set the floor to 1.

Then the player will be led to the main menu where the player can choose to view the scoreboard list in different order, search for a player and the player's information, enter the game, and quit the game and save the information.

After entering the game, the program will ask the player the difficulty he or she likes to play. If the player enters 'E', then the difficulty modifier will be set to 1. Similarly, if the player enters 'M' or 'D', the modifier will be set to 2 and 3.

In the next instructions, the program will put a bunch of enemies into an enemy queue. These enemies have different strengths. The later enemies are more powerful. Each time a combat starts, an enemy will be called from the queue and fight against the player. After

the combat with the player ends, the enemy will be removed from the queue. Then, some weapons will be added to the shop and some other weapons will be added to the shop later on based on the floor.

At this point, the game will officially start using a while loop. In each iteration, the program will check if the player has reached the 4th floor. If it is on the 5th floor, then the program will end. It will save the information and print out the game information (ie. player scores, enemy defeated, completed side quests, etc.). The program also distributes different side quests and adds new weapons to the shop when the player reaches different floors. The PlayerOption function will first be called in another while loop. The while loop will break until the output of the playerOption is not 0. After that while loop breaks, the enemyCombat and PlayerCombat functions are called. With the two functions, the player and the enemy will fight in turns until one side's current health drops to 0. After the enemy is defeated, the program will check if there is any side quest completed by checking the global variables that are related to the certain side quests. If the side quest is completed, then extra score will be added to the scoreboard. After that, the player 's current health is set to be equal to max HP and moves to the next floor. The new iteration starts.