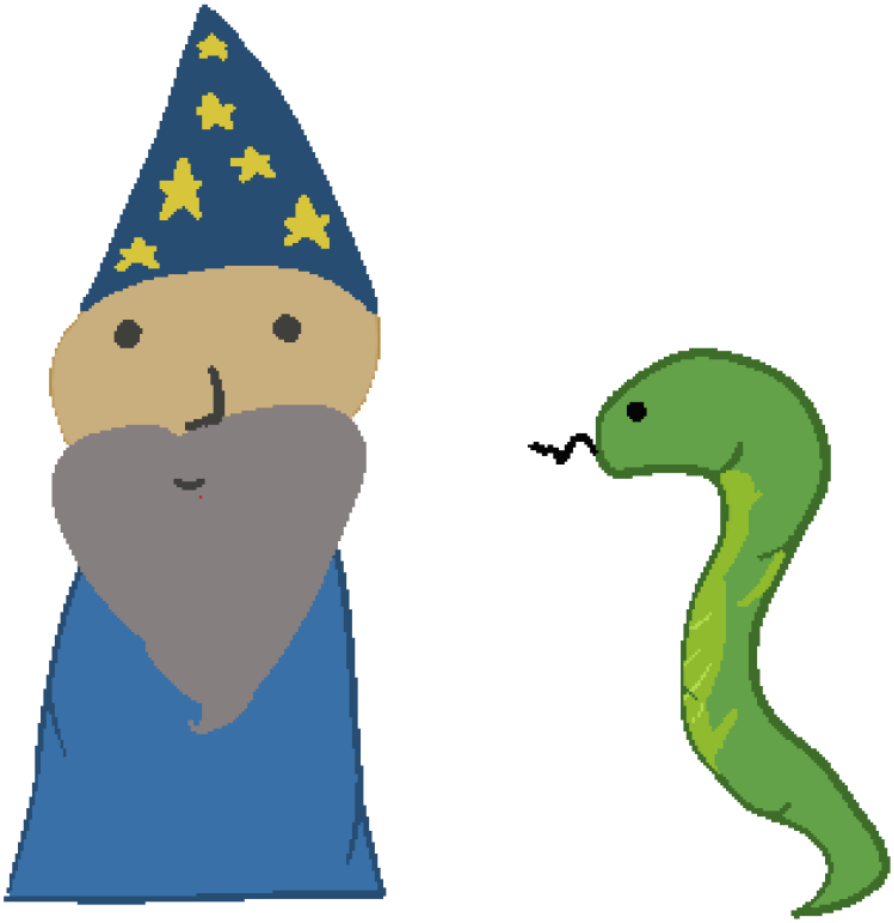
Project 03
Wizards and Lizards
An upcoming Role Playing Game



Project Overview

Background

It is the year 1995, you have recently gotten home to your computer after a long session of Dungeons and Dragons with your friends. The session was very amazing to the point that you wanted to recreate the game.

As a very simplistic game, the game length will consist of a random number of battles that will happen in succession. The player will first have to create a number of characters, and the system will randomize a list of enemies that the player has to fight against. Thankfully, you were able to implement the character creation, but now are tasked in implementing the rest of the game systems.

The game consists of three systems:

- **The Ability System:** This will hold all the information pertaining to abilities used by players and enemies. You will be storing all abilities in a “database” class, for easy access and managing memory regarding abilities.
- **The Entity system:** This system is in charge of storing the information of players, what abilities they have, and other useful information pertaining to character interaction.
- **The Battle system:** This system will focus on combining the entity system and the ability system into a cohesive and functioning game. It will manage the combat flow of players, giving them a chance to use abilities that are influenced by entity stats.

As such, the project is split into three phases:

- **Phase 01: Setting up an Ability Database**
    - This phase focuses on reading data from external files, dealing with dynamic memory, and basic implementation of functions within classes.
- **Phase 02: Entity System**
    - This phase will teach you how to deal with inheritance, overloading functions within classes, and setting up some necessary implementations for the third phase.
- **Phase 03: Battle System**
    - This phase will handle the logic and interaction of different systems created from previous phases. It will also manage the erasing of dynamic memory.

Important Information:

You must be on top of this project which is why three lab sections are dedicated for this project. **Questions regarding the project will only be answered during office hours, lab hours, lectures, or through email.**

Release Dates:

Phase 01 will be released July 6th 2021.
Phase 02 and 03 will be released July 12, 2021 at the latest.

Due dates:

Phase 01: July 12 → 11:59 PM EST
Phase 02: July 19 → 11:59 PM EST
Phase 03: July 23 → 11:59 PM EST (NO LATE SUBMISSIONS)

If any circumstances come up, EMAIL ME AS SOON AS POSSIBLE. The end of the semester is July 26th, as well as your final grades. Consider this as a professional assignment at a workplace, the product needs to go into "deployment."

Phase 01: The Ability System

---

01. Overview

For this phase, you are in charge of creating an “Ability Database.” You will be in charge of designing two classes: Ability and AbilityDB. The AbilityDB stores a collection of dynamically created Ability objects. You would be accessing Abilities within the AbilityDB class using pointers, similar to how you would access within a database.

The TSV file can be downloaded here:
https://docs.google.com/spreadsheets/d/1z64dldukPDbnyzgq0As91I-vKLJXMQENiF87RrxUemg/edit?usp=sharing

02. Ability Class

The public methods for the `Ability` class are given to you in this documentation. The class represents an individual ability. A tsv file is provided that contains all the fields that need to be stored in the ability class. The Ability class encapsulates the following data set.

Data Dictionary

- `string name;` the name of the ability, it could include punctuation, space, and other characters with the exception of ‘\t’. Example “Ram’s Voice”
- `int damage;` a non-negative integer that stores the damage an ability deals.
- `int cost;` an integer that stores the cost of an ability.
- `DamageType type;` The type of damage it deals, could be “physical”, “magical”, or “healing”
- `Job job_specific;` Should be either: “warrior” “cleric”, “bard”,“wizard”, or “monster”

These are private data members of the Ability class. Make sure the syntax is identical to everything labeled in this project.

Method Syntax	Description
Ability(string n, int d, int c, DamageType dt, Job j);	A constructor that takes in a name, damage, cost, damage type, and job.
Ability(const string & tsv_line);	A constructor for the class that takes a string from a tsv file.
string getName() const;	This returns the name of the Ability
int getCost() const;	This returns the cost of the Ability
int getDamage() const;	This returns the damage of the Ability.
DamageType getDamageType() const;	This returns the damageType of the Ability.
Job getJob() const;	This returns the Job of the Ability.
int calculateDamage(int attack, int defense);	This returns a calculation of the damage from the ability.

About int calculateDamage(int attack, int defense);

Damage is done additive, meaning that if a user were to take damage, it should return a positive value, and if they are being healed, should return a negative value. If the damageType of the ability is healing, it should ignore defense. Otherwise, it needs to calculate the sum of damage and attack, and then subtract from defense.

If the user is not healing but the actual damage dealt results in a negative damage (being healed), it should return 0.

03. AbilityDB Class

The public methods for the `AbilityDB` class are given to you in this documentation. The class represents a collection of abilities. The tsv file provided contains a list of abilities that the AbilityDB will read, and dynamically create all abilities. This object only stores one variable:

- `vector<Ability*> abilities;` Stores all the Ability objects.

Method Syntax	Description
AbilityDB(string tsv_file);	A constructor for the class that receives the path to a .tsv file that contains all the abilities.
Ability* getAbilityByID(int id);	Returns an Ability* of the specified ID. In this case, the ID is the index of the vector.
Ability* getAbilityByName(string name);	Returns an Ability* with the specified name.
vector<Ability*> getAbilitiesByClass(Job job);	Returns a vector of Ability* with the specified Job.

04. Files

A utils.hpp and utils.cpp will be provided. The files can be downloaded here. These contain the enumeration for DamageType and Job, as well as methods that convert them to the proper enumeration. These methods are helpful.

You are required to submit the following files for this phase of the project:

- Ability.hpp
- Ability.cpp
- AbilityDB.hpp
- AbilityDB.cpp

When working with these files you must remember the following:

- The CPP file needs to include the HPP files. If a file requires another class, include the HPP file, not the CPP file.
- You need to include header guards. You can look at utils.hpp for reference. (#ifndef, #define, and #endif).
- ONLY compile CPP files. NEVER compile the HPP files.

Phase 02: The Entity System

---

01. Overview

At this point, you should have reached completion in setting up an Ability Database that stores all the abilities from a .tsv file. Now comes the second phase, **adding enemies and players to the game**.

You will be focusing on making a base class called `Entity`, which handles statistics and critical information of both a player and enemy. After you have made this base class, you will create two derived classes called `Player` and `Enemy`, which have their own specifications that build up on the Entity class.

02. Entity Class

The entity class is your base class. The documentation is outlined similarly to phase01 and shouldn't be that hard to manage.

Data Dictionary

- `string name_;`
- `int stats[7];` the statistics of the Entity. The data is separated as {MaxHP, MaxSP, Attack, Magic, Defense, MagicDefense, Speed}
- `int current[2];` Contains the {CurrentHP, and CurrentSP} of the Entity.

These are private data members of the Entity class. Make sure the syntax is identical to everything labeled in this project.

Method Syntax	Description
Entity();	A base constructor class. Should set all values to empty string or 0.
int getMaxHP() const;	Returns MaxHP.
int getMaxSP() const;	Returns MaxSP.
int getAttackstat(DamageType stat) const;	Returns either the Attack stat or Magic stat based on the DamageType specified.

int getDefenseStat(DamageType stat) const;	Returns either the Defense or MagicDefense stat based on the DamageType specified.
int getSpeed() const;	Returns speed value.
int getCurrentHP() const;	Returns CurrentHP.
int getCurrentSP() const;	Returns CurrentSP.
void setName(string name);	Sets the name of the entity.
void setStat(int id, int value);	Based on the position of the stat, should change it to set value.
void setStat(vector<int> values);	Receives a vector of stats that would modify the stat. If the value is -1, it should ignore that stat when setting the value. The vector size could also be less than the stats array.
void setCurrentHP(int value);	Sets currentHP to value.
void setCurrentSP(int value);	Sets currentHP to SP.

03. Player Class

The Player Class will be your first derived class. The Player Class will be used by an external user. Therefore, there needs to be an increased level of intractability with Player Objects.

Data Dictionary

- `Job job; The Job the player belongs to.`
- `int level; The level of the Player`
- `int currentXP;` Stores how much experience Player is currently holding.
- `int nextLevel];` Stores how much experience is needed until the Player levels up.
- `vector<int> abilities;` Stores all the ability IDs the Player has. By default, the vector should have all the skills pertaining to that Job.

Method Syntax	Description
Player(string n, vector<int> st, Job j);	Parameterized constructor for the player class. Contains name, stats, and which job the Player is.
vector<Ability> listAbilities();	Returns a vector of IDs belonging to the skills.
int getCurrentXP() const;	Returns currentXP.
int getnextLevelXP() const;	Returns nextLevel;
bool gainXP(int value);	Adds xp to currentXP. If they level up, currentXP and nextLevel should be updated to reflect. Returns true if the Player has leveled up.
void attackEnemyWithSkill(Enemy & target, int id);	Uses specified Ability using its ID from the AbilityDB on target Enemy. If the ID is -1 or not an ability “learned” by the player or valid, it should perform a “normal” attack, where it decrements the target's health by (attack - defense). Like phase01 calculateDamage, if the damage < 0, it should not decrement health.
void attackPlayerWithSkill(Player & target, int id);	Uses specified Ability using its ID from the AbilityDB on target Player.If the ID is -1 or not an ability “learned” by the player or valid, it should perform a “normal” attack, where it decrements the target's health by (attack - defense). Like phase01 calculateDamage, if the damage < 0, it should not decrement health.

NOTE: Bolded Methods will not be tested for Phase02, but will be useful for Phase03.

04. Enemy Class

The Enemy Class is your second derived class. As this will not be controlled by the user, it does not have as many methods as the Player Class.

Data Dictionary

- `int XP;` Contains the XP that Enem
- `vector<int> abilities;` Stores all the ability IDs the Enemy has. By default, the vector should have any skills belonging to “monster”.

Method Syntax	Description
Enemy(string n, vector<int> st, int xp);	Parameterized constructor for the Enemy class. Similar to player, it contains the name, stats, and a set value of xp.
int getXP();	Returns the value for XP.
void attackPlayerWithSkill(Player & target, int id);	Uses specified Ability using its ID from the AbilityDB on target Player. If the ID is -1 or not an ability “learned” by the player or valid, it should perform a “normal” attack, where it decrements the target's health by (attack - defense). Like phase01 calculateDamage, if the damage < 0, it should not decrement health.

NOTE: Bolded Methods will not be tested for Phase02, but will be useful for Phase03.

05. Files

A utils.hpp and utils.cpp will be provided. The files can be downloaded here. These contain the enumeration for DamageType and Job, as well as methods that convert them to the proper enumeration. These methods are helpful. Assuming Phase01 is correctly implemented, you should be using

You are required to submit the following files for this phase of the project:

- Entity.hpp
- Entity.cpp
- Player.hpp
- Player.cpp
- Enemy.hpp
- Enemy.cpp

When working with these files you must remember the following:

- The CPP file needs to include the HPP files. If a file requires another class, include the HPP file, not the CPP file.

- You need to include header guards. You can look at utils.hpp for reference. (#ifndef, #define, and #endif).
- ONLY compile CPP files. NEVER compile the HPP files.

Phase 03: The Battle Manager

---

01. Overview

For this final phase, you are in charge of designing and implementing a BattleManager Class. You will be responsible for the methods, data dictionary, and implementation of this class. However, there are some aspects that are more-or-less required to make a successful combat simulation:

- **Display:** You need to display the information of the battle which may include: Health of Players, skills that a Player can use, and other useful information.
- **User Input:** A user should be able properly input information to the Player objects, and should be responsive.
- **Interaction between objects**: Objects should be able to interact with each other and there should be

The main objective of this phase, besides adding interactions between multiple systems, is to test on your **own** personal design choices. Software Engineering is not just about the data, but is also about how the data is shown. You are required to know both of these aspects in order to create an engaging product that is useful for fellow engineers and the user.

Since you are given freedom to do as you please for this Class, you are free to **modify and add** methods from the other classes that were done in the previous methods. Removing methods may be allowed, as long as it doesn't change the "core" aspect of those Classes. Take this opportunity to let your creativity and knowledge in coding expand on this project to something that is uniquely yours.

Resources:
Turn-Based combat : This is the type of combat that I am looking towards, as it is easier to implement.

02. Phase03 Grading

Since this project is essentially a game, I will be evaluating this phase using the MDA framework. This is defined as followed:

- **Mechanics**: This is an aspect worked on the previous two phases. This includes the algorithms and data structures. You are free to expand on the current data structures, create more data structures to properly organize your data.
- **Dynamics:** This aspect is how the game interacts with other systems during runtime.

- **Aesthetics:** This is how the game "feels," what type of experience you are trying to convey within the game. In other words, how you display the data to the user. Even though the article talked about eight basic types. The mechanics and dynamics of your code will help properly convey the type of aesthetic you are trying to convey.

For More information, feel free to read the journal article on MDA approach here.

03. Files

You need to upload ALL the files. This includes a functional main.cpp that your program will run. This means that the following need to be sent:

- Phase01 Files
  - Ability.cpp
  - Ability.hpp
  - AbilityDB.cpp
  - AbilityDB.hpp
- Phase02 Files
  - Entity.hpp
  - Entity.cpp
  - Player.hpp
  - Player.cpp
  - Enemy.hpp
  - Enemy.cpp
- main.cpp
- BattleManager.hpp
- BattleManager.cpp
- Any other files that you decided to create.

This seems like a lot of files to both submit and to compile. This is why you are require to do the following:

- Send a ZIP file of all your code.
- To better compile all your files, I recommend that you read up on "make" files and have one when testing your code out. Many of you are running your code on Windows systems. You might have to do some research on this, but here is a start to your searches:
  - https://opensource.com/article/18/8/what-how-makefile
  - https://stackoverflow.com/questions/54742271/how-to-compile-code-with-a-makefile-in-windows-command-prompt-using-mingw-compil