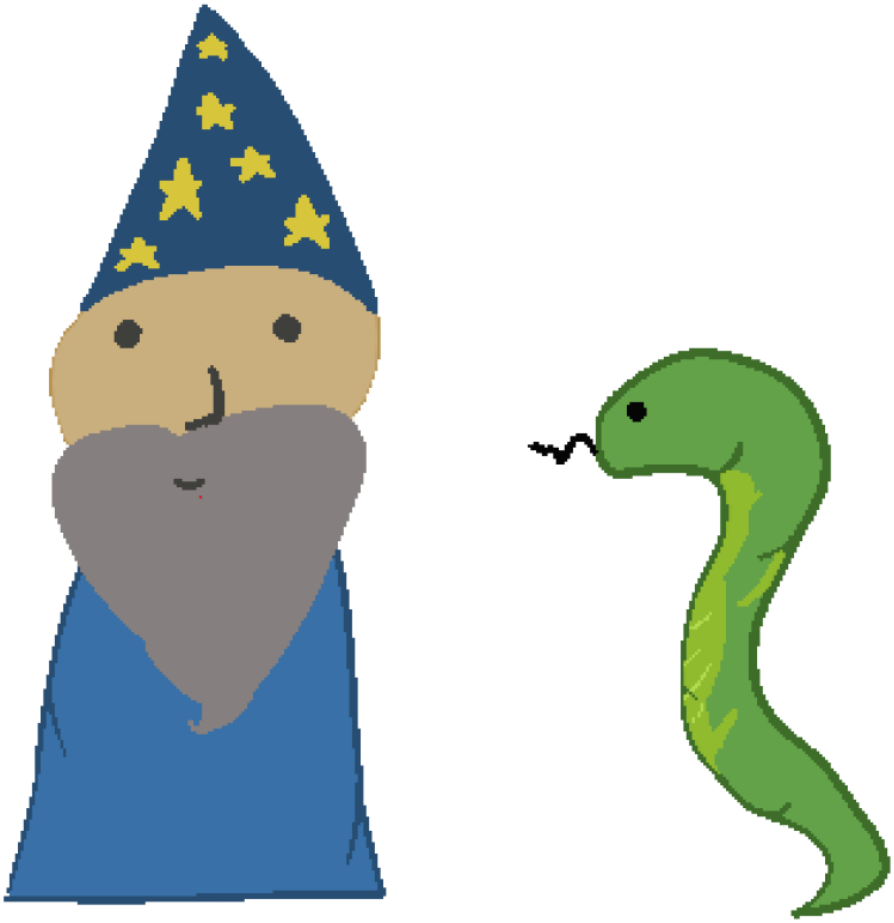
Project 03
Wizards and Lizards
An upcoming Role Playing Game



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.