OOP Project Design Report (Draft)

Student: Fuad Ismayilbayli
Project Title: RPG Battle Simulator(actually I still smth from BG3)

1. Introduction

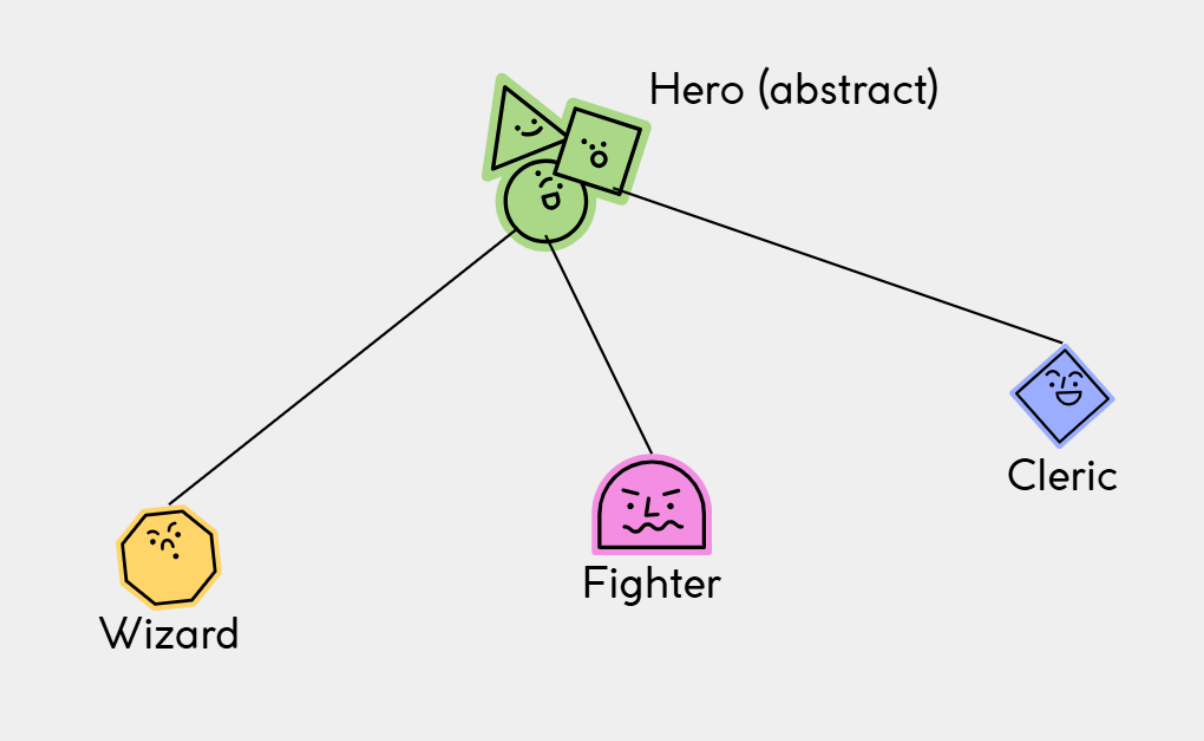
This project is a simple RPG battle simulator inspired by *Baldur's Gate 3*.
It allows the user to create heroes of different **classes** (for example, Wizard, Fighter, or Cleric) and **races** (such as Human, Elf, or Tiefling).
Each hero has unique health, damage, and a special ability.
The program simulates a short fight between two heroes and shows how their attacks and healing work.

The goal of this project is to demonstrate **Object-Oriented Programming (OOP)** in Python by using classes, inheritance, and polymorphism.
It helps to understand how complex game systems can be organized into objects with clear structure and behavior.

The project is written in **Python** using only the **standard library** (abc for abstract classes and unittest for testing).
It runs in the console and does not need any external modules.

2. Class Design

The project includes several classes that represent game entities.
The main structure is shown below:



Class	Description	Main Attributes	Main Methods
Hero	Abstract base class. Defines shared attributes and actions for all heroes.	name, damage, health, max_health	hit(), heal(), set_damage(), use_special() (abstract)
Wizard	Inherits from Hero. Has lower health but strong magic attack.	Inherits all	use_special() – casts a fireball with extra damage.
Fighter	Inherits from Hero. Strong defense and melee attack.	Inherits all	use_special() – performs a powerful strike.
Cleric	Inherits from Hero. Can heal more effectively.	Inherits all	use_special() – restores health using divine magic.

Each hero also belongs to a **race** (Human, Elf, Tiefling, etc.), which gives small bonuses to health or damage.
Race bonuses are stored in a dictionary for easy access.

Design decisions:

- The abstract class Hero helps to avoid repeating code.
- Each subclass implements its own special attack using **method overriding**.
- Attributes are not changed directly from outside — they are controlled through methods like hit() and heal().

3. Application of OOP Principles

The project applies all four main principles of OOP:

1. Encapsulation

All hero data (health, damage) is stored inside the class.
It can only be changed using specific methods such as hit() or heal().
This protects the internal state of objects.

```
def hit(self, target):
    target.health -= self.damage
```

2. Inheritance

All character types (Wizard, Fighter, Cleric) inherit from the base class Hero.
This allows them to reuse common attributes and methods.

```
class Fighter(Hero):
    ...
```

3. Polymorphism

Different hero classes share the same method name use_special(), but each one behaves differently.
This allows the same interface for different actions.

```
hero.use_special(enemy)
```

4. Abstraction

The class Hero is declared as **abstract** using the ABC module.
It defines the interface that all heroes must follow.

```
from abc import ABC, abstractmethod
class Hero(ABC):
    @abstractmethod
    def use_special(self, target):
        pass
```

This makes the code more structured and easier to extend with new hero types.

Summary:

The project demonstrates how OOP helps to organize game logic clearly.
It shows encapsulation (data hiding), inheritance (shared behavior), polymorphism (different abilities), and abstraction (base design).
Future improvements could include adding more hero classes, items, or a turn-based combat system.