## SOLID Principles Application

This project applies SOLID principles to make the code more understandable, flexible, and easier to extend. Here's how each principle is implemented:

### 1. Single Responsibility Principle (SRP)

Each class is responsible for a single task:

- **Player** handles the player's health, experience, and inventory.
- **CombatManager** manages the combat mechanics.
- **ItemManager** handles items that the player can pick up.
- **LevelManager** tracks levels and enemies on each level.
- **ScoreManager** displays the player's score.

This approach makes the code easier to maintain and read.

### 2. Open/Closed Principle (OCP)

Classes are open for extension but closed for modification. For example, if we want to add a new enemy or item, we don't need to change the existing classes. We simply create a new class that uses the existing interfaces (such as **Enemy**), and everything works as expected.

### 3. Liskov Substitution Principle (LSP)

We can substitute one class for another without breaking the program. For example, all enemy classes (like **Zombie**, **Skeleton**, **Creeper**) implement the **Enemy** interface. This means any class that implements this interface can be used interchangeably without modifying the combat system.

### 4. Interface Segregation Principle (ISP)

Each interface contains only the methods that are necessary for a particular role. For example, the **Enemy** interface only includes methods relevant to enemies (like **getDamage())**, avoiding unnecessary functionality. This makes the code more lightweight and modular.

### 5. Dependency Inversion Principle (DIP)

High-level modules don't depend on low-level modules, but both depend on abstractions. For instance, **CombatManager** doesn't rely on specific enemy types but works with the **Enemy** interface. This allows us to easily add new enemies or change combat logic without affecting other parts of the code.