1.For Spawner class:
SRP: Single Responsibility Principle
Spawner, as an ADT class, defines the spawning of creatures. The class only defines the spawning logic, without specifying the spawning probability (this.chance = chance). The specific spawning probability is completed by the child system.

OCP: Open/Closed Principle
An abstract spawn() method is defined, and subclasses can extend the spawning of different types of characters by implementing this method without modifying the code of the Spawner class. ManFlySpawner and SpiritSpawner extend Spawner to provide specific spawning logic for different creatures without modifying the code of the Spawner itself.

ISP: Interface Segregation Principle
The Spawner here only requires subclasses to implement the spawn() method to handle the spawning logic. There is no interface in Spawner, and subclasses are not forced to implement other redundant methods, thus complying with the Interface Segregation Principle.

DIP: Dependency Inversion Principle
In this example, the Spawner class depends on the abstract Actor rather than a specific character class. Specific mob spawning logic, such as ManFly or Spirit, is implemented in subclasses rather than having a direct dependency on a specific character class in Spawner.

2. For SpiritSpawner class:
SRP: Single Responsibility Principle
The only responsibility of the SpiritSpawner class is to spawn Spirit characters. It provides the logic for creating Spirit instances by implementing the spawn() method. In addition to spawning Spirit, it does not handle any other functions, nor does it contain logic unrelated to spawning other characters, which ensures the singleness of responsibility. The SpiritSpawner class only handles the logic of spawning Spirit, and other responsibilities, such as character status management or display functions, are not included, in line with the single responsibility principle.

OCP: Open/Closed Principle
SpiritSpawner inherits from Spawner and extends the specific spawning logic by overriding the spawn() method. The Spawner class itself does not need to be modified to extend the new spawning character type through inheritance. In the future, if there are new character types that need to be spawned, we only need to create a new spawning class that inherits Spawner, such as ManFlySpawner or other character spawning classes, without having to modify the code of SpiritSpawner itself. The ManFlySpawner class is an extension of Spawner, similar to SpiritSpawner, which provides a specific implementation for spawning ManFly characters. This shows that the Spawner class follows the open-closed principle, and subclasses add new functionality by extending rather than modifying the parent class.

LSP: Liskov Substitution Principle
The addSpawner() method can accept a Spawner type argument and can use SpiritSpawner and ManFlySpawner instead without changing the calling code. This ensures that SpiritSpawner follows the Liskov Substitution Principle.

ISP: Interface Segregation Principle
Spawner only provides the minimum functional interface for spawning characters, and SpiritSpawner only needs to implement this interface without any additional dependencies. This complies with the interface isolation principle, ensuring that classes only need to implement the interfaces they care about.

DIP: Dependency Inversion Principle
The SpiritSpawner class depends on the abstract Actor class, not a specific character implementation. It returns the abstract type Actor through the spawn() method, rather than relying on the details of a specific implementation. In this way, if the implementation details of the Actor class need to be modified, it will not affect the logic of SpiritSpawner, following the Dependency Inversion Principle.

3. For ManFlySpawner class
SRP: Single Responsibility Principle
The ManFlySpawner class is responsible for spawning ManFly creatures. It implements the spawn() method to generate the specific logic of ManFly. Other functions not related to spawning ManFly are not included in this class. This ensures a single responsibility. It only includes setting the probability of spawning ManFly to 15% and spawning ManFly instances.

OCP: Open/Closed Principle
The ManFlySpawner class extends the logic of spawning ManFly by inheriting the Spawner class. The Spawner class provides an abstract framework for spawning creatures, and ManFlySpawner implements the specific spawning logic by extending Spawner. This shows that the Spawner class is open to extension but closed to modification. If there are new creatures to spawn later, just create a new creature class without modifying the existing Spawner class or ManFlySpawner class.

ISP: Interface Segregation Principle
ManFlySpawner only needs to implement the necessary interfaces defined in the Spawner class (such as the spawn() method). It is not forced to implement unnecessary interfaces, and functions unrelated to the spawning logic are isolated to other classes. Therefore, ManFlySpawner follows the interface segregation principle. The Spawner class only defines the minimum interface related to biological generation, and ManFlySpawner only needs to implement the spawn() method. This shows that ManFlySpawner does not rely on unnecessary interfaces and complies with the interface isolation principle.

DIP: Dependency Inversion Principle
The ManFlySpawner class depends on the Actor abstraction, not the specific biological implementation. By implementing the spawn() method, ManFlySpawner returns an abstract Actor object without depending on the specific ManFly implementation. Only relying on the abstract type of Actor, without caring about the internal details of ManFly, conforms to the Dependency Inversion Principle.

4. For Graveyard class:
SRP: Single Responsibility Principle
The Graveyard class represents the graveyard terrain and manages the spawning of creatures on it. It spawns creatures by combining Spawner, but is not responsible for other

complex logic, such as creature state management or attack. This ensures that the Graveyard class has a very clear responsibility: controlling the spawning logic in the graveyard, not other game features.

DIP: Dependency Inversion Principle
The Graveyard class depends on the abstract Spawner instead of the specific SpiritSpawner or ManFlySpawner implementations. By relying on the Spawner abstract class, the Graveyard can spawn different creatures based on different Spawner implementations without modifying its own logic. In this way, if a new creature needs to be spawned, only a new Spawner implementation needs to be created without affecting the Graveyard. This shows that the Graveyard follows the Dependency Inversion Principle.

5. For Spirit class
SRP: Single Responsibility Principle
The Spirit class represents the Spirit creature character in the game. It is only responsible for storing the Spirit's attributes (name, char, hitpoints, weapons). Generation logic and character status updates are not within its scope of responsibility, ensuring a single responsibility.

OCP: Open/Closed Principle
The ManFly class, like the Spirit class, inherits from Enemy and extends its specific functions. Through inheritance, the functions of enemy types can be extended without modifying the code of the Spirit class, which conforms to the open-closed principle. The Spirit class extends its functions by inheriting the Enemy class. Enemy is an abstract class or base class that defines the common behaviors and properties of all enemies, while Spirit implements its specific properties and behaviors by inheriting Enemy.