Análisis del uso de Patrones de Diseño

- **Visitor:** permite separar el algoritmo de la estructura de datos que se utilizará para ejecutarlo. Este patrón favorece la incorporación de nuevas operaciones a estas estructuras sin necesidad de modificarlas.
  Se utiliza Visitor en el mecanismo encargado de detectar colisiones entre entidades. Justamente lo que se pretende es que las entidades colisionen entre sí sin necesidad de saber desde un principio de que tipo es cada entidad que esta colisionando. Además, utilizando este patrón se favorece que en un futuro se puedan agregar otras entidades al Juego sin necesidad de modificar demasiado código.

- **Strategy:** permite que el algoritmo varíe independientemente de los clientes que lo usan. Entre uno de sus beneficios podemos destacar que reduce múltiples sentencias condicionales en la clase cliente.
  Se utiliza Strategy para manejar la Inteligencia que posee cada Entidad. A lo largo del juego esta inteligencia puede cambiar, como por ejemplo en el caso de algunos enemigos, y sin embargo no se deberá modificar el código por este cambio.

- **Abstract Factory:** nos provee una interfaz que delega la creación de un conjunto de objetos relacionados sin necesidad de especificar en ningún momento cuáles son las implementaciones concretas.
  Se utiliza Strategy para la creación de Armas. En este caso la Abstract Factory seria la clase abstracta Arma, la cual posee un método que crea disparos y luego cada clase concreta de Arma se encargara de crear su propio disparo con características específicas.

- **Decorator:** permite añadir funcionalidad extra a un objeto (de forma dinámica o estática) sin modificar el comportamiento del resto de objetos del mismo tipo.
  Utilizaremos Decorator para modelar la acumulación de PowerUps que afectan al Jugador, ya que este mismo puede estar afectado por varios PowerUps en un mismo momento.