Hemos incorporado algunos patrones de diseño al juego, como pueden ser el “Singleton”, “Visitor” y pensamos incorporar los patrones “State”, “Abstract Factory”, “Decorator”.

El patrón “Singleton” lo utilizamos en clases que necesariamente se crean solo una vez. Esto asegura que el acceso a la instancia única esté controlado. También se reduce el espacio de nombres frente al uso de variables globales.
Lo aplicamos en las clases del jugador, los niveles y las inteligencias.

El patrón “Visitor” lo utilizamos para detectar las colisiones entre los objetos en el mapa del juego. En éste, es fácil añadir nuevas operaciones y/o detectar nuevas colisiones, y brinda la posibilidad de, en caso de añadirse nuevos objetos al juego, crear un nuevo visitante y realizar muy pocos cambios en el código que tenemos, para detectar colisiones o interacciones entre el objeto nuevo y los ya existentes.

El patrón “State” lo utilizaremos para determinar el comportamiento, de manera dinámica, de los enemigos. Permite una gran flexibilidad para añadir nuevos estados y transiciones, simplemente bastará con definir un nuevo comportamiento, y no se modificará el código que tenemos.

El patrón “Abstract Factory” lo utilizaremos para crear los objetos del mapa. Ayuda a mejorar el encapsulamiento, debido a que se aislá a los clientes de las implementaciones. También se incrementa la flexibilidad del diseño, permitiendo cambiar factiblemente de familias de productos.

El patrón “Decorator” lo utilizaremos para modelar el sistema de Power Up del juego. Provee una alternativa a la herencia estática, permitiendo añadir funcionalidades de manera mas flexible. Y también así evita la concentración en lo alto de la jerarquía de clases “guiadas por las responsabilidades”, es decir, que pretenden satisfacer todas las posibilidades.