Justificación de decisiones de diseño

En el main se usa una especie de método fabricante porque la clase *Controller* llama al método *startGame*, este método se encarga de crear las instancias de los diferentes objetos que se necesitan para el juego.

El patrón fábrica también se implementa en la creación de las piezas. El método *makePiece* de la clase *PieceFactory* es el encargado de crear los diferentes tipos de piezas, y estas se diferencian por los movimientos válidos que poseen, definidos desde la creación de la pieza.

La decisión de usar este patrón es porque sirve para separar el código de construcción de las piezas del código que usa las piezas. Esto permite de manera más fácil extender el código para poder construir piezas de forma independiente al resto del código.

Las ventajas de utilizar el patrón de fábrica es que se evita un acoplamiento fuerte entre la parte que crea las piezas y las clases que representan los tipos concretos de cada pieza.

Al usar este patrón también se aplica el principio de responsabilidad única y el principio de abierto/cerrado porque se pueden agregar nuevos tipos de piezas en el programa sin modificar el código que ya existe.

Por otro lado, también se utilizaron clases abstractas para el tablero lógico, la interfaz gráfica, el controlador, el árbitro, el administrador de archivos (para el guardado y cargado de partidas), el jugador, y por último las piezas, lo que implica que estas hacen uso de métodos plantilla a la vez que son creadas por la fábrica anteriormente mencionada. Los métodos plantilla se idearon con el fin de que funcionen como una guía y se puedan extender a otros juegos de la misma familia, de esta forma, si se quiere añadir más clases para otros juegos, se puede hacer más sencillo, ya que en las clases abstractas se tiene parte del comportamiento que deben tener, y solo se tiene que implementar los atributos y métodos específicos de cada clase. Con esto implícitamente se aplica el principio de sustitución de Liskov porque los objetos pueden ser reemplazados por instancias de sus subtipos sin cambiar el funcionamiento del juego, es decir, se puede usar cualquiera de las subclases sin interferir la funcionalidad del programa.