Detalles de implementación MARDA

Diseño de Software

Denisse Alfaro B50220
Roberto Vargas B57617
Aaron Gutierrez B63241

I Semestre 2019

A lo largo del desarrollo del MARDA se implementaron distintos patrones tanto de diseño como de arquitectura que facilitaron el mantenimiento y la extensión del código. El patrón de arquitectura usado fue MVC, Modelo Vista Controlador el cual es uno de los más importantes ya que funciona como un intermediario entre la vista y el modelo, permitiendo actualizar la interfaz gráfica de acuerdo a los eventos que suceden con respecto a la información que brinda el modelo, haciendo así que la interacción con el Framework sea posible.

Además de un patrón arquitectónico, se aplicaron patrones de diseño como lo fue el patrón de composición, patrón que implícitamente se aplica en MVC al utilizar el lenguaje Python, esto debido a que el controlador necesita estar compuesto por una instancia de la vista y cuantas instancias sean necesarias del modelo para realizar la interacción pertinente. También se usó el patrón puente que permite combinar abstracciones e implementaciones diferentes y extenderlas independientemente, haciendo así que existan métodos plantilla los cuales pueden ser usados por las extensiones concretas de las clases como por ejemplo en la clase Jugador, la clase Pieza o la clase Arbitro.

También se aplicaron patrones de comportamiento como Estrategia y Observador. Ambos patrones vienen incluidos en el patrón de arquitectura Modelo-Vista-Controlador cuando es aplicado de manera correcta. Se aplica el patrón Estrategia ya que el controlador le da un comportamiento específico a la vista del programa, si se desea, se puede crear una implementación distinta, que genere los mismos resultados o bien resultados visualmente distintos. Así como en los libros se explica estrategia con algoritmos de ordenamiento, existen muchos algoritmos que hacen lo mismo, pero de maneras distintas, cada una con sus ventajas.

En cuanto al patrón observador, se sabe que este define una dependencia del tipo uno a muchos entre objetos y cuando un objeto actualiza su estado, notifica de ese cambio a los objetos dependientes. En el Modelo-Vista-Controlador, el modelo actúa como un observador y cuando este se actualiza, hace que tanto la vista como el controlador se enteren de los cambios, por tanto, se aplica de manera correcta este patrón de comportamiento.

Bien se sabe que para que exista código limpio y bien hecho se debe hacer uso de los tan importantes principios SOLID y el MARDA no es la excepción. Uno de los principios presentes en este Framework es el Single Responsability esto quiere decir que no solo cada una de las clases tiene una única responsabilidad sino también los métodos, haciendo así que el mantenimiento del código sea más fácil. Relacionado a esto es que se tomó una decisión importante en la implementación y fue la creación de una clase árbitro que es la encargada de toda la validación y reglamentación del juego, quitando así esta responsabilidad al tablero el cual la tenía anteriormente.

Ahora hablando del principio Open-Closed, esté se refiere a que un programa esté abierto a la extensión y cerrado al cambio. Es decir, la capacidad de añadir funcionalidades sin modificar la estructura actual del programa. En nuestro MARDA este principio se cumple también gracias al uso de clases abstractas. Éstas permiten cambiar la clase concreta sin modificar las interfaces. Lo anterior se ve demostrado en las clases abstractas que nuestro MARDA posee, I_Jugador, I_Tablero, I_Vista sólo para mencionar algunas. Además, para cada una de ellas se incluyen las clases concretas que implementan los métodos ahí especificados (Jugador, Tablero, Vista, etc).

Se aplica además el principio Liskov Substitution gracias al uso correcto de la herencia. Esto porque permite que una instancia de alguna clase derivada pueda reemplazar una instancia de una clase padre y que el programa continúe con un correcto funcionamiento. Un ejemplo puede ser reemplazar una instancia de Tablero por una instancia de Tablero Go. Así queda demostrado que el cambio de instancias no va a afectar el funcionamiento general del framework.

Finalmente, Dependency Inversion se ve aplicado en nuestro MARDA también. Nuestras clases principales no son dependientes de clases más específicas. Por ejemplo, el agregar métodos específicos del juego de mesa Go en nuestra clase Jugador Go no afectan para nada a la clase general Jugador. Además, se podría decir que esté principio se ve aplicado casi por default una vez que Open-Closed y Liskov Substitution son usados.