Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

CI-0136 Diseño de Software

Tercera Iteración: MARDA

Profesor Alan Calderón Castro

Integrantes:

Daniel Monge Arce B85017
André Villegas Gallardo B98556
Alejandro Sanchez Fong B66590
Fernando Agüero Cordero B60086

I Ciclo 2022

Justificación de las principales decisiones de diseño adoptadas por el equipo

- **Clase abstracta Deck**

Se crea una clase abstracta 'Deck' para poder facilitar la creación de distintos tipos de mazos de diferentes tipos, lo cual permite el uso de estos en otros juegos que lo requieran. En el caso de Durak se utiliza un mazo con cartas por jugador.

- **Clase Serializer**

Se tomó la decisión de crear una clase que permita guardar y cargar la partida de diferentes juegos a partir de un serializador abstracto, esto con la idea de tener más código genérico y permitir la adición de serializadores distintos para guardar y cargar el estado del juego.

- **Clase Card**

Una de las decisiones tomadas fué hacer una clase 'Card' que permite la construcción de cartas, que son llamadas por la clase abstracta 'Deck', además de poder establecer y obtener los atributos de cada carta, permite que está sea reutilizable.

Patrones Aplicados

- **Arquitectura Modelo-Vista-Controlador (MVC)**

Utilizamos el patrón de arquitectura MVC ya que nos facilita la separación del juego en tres capas, es conveniente de utilizar para juegos de mesa ya que estos necesitan de algún rol controlador, una vista para que los usuarios jueguen y modelos que vienen a ser las abstracciones de cartas o jugadores.

- **Método plantilla**

Utilizamos el patrón de comportamiento plantilla para darle un esqueleto al juego 'Durak' mediante la clase abstracta *'Game'*, así es más fácil implementar algún otro juego de cartas distinto.

- **Patrón Constructor**

La clase *'Abstract_Serializer'* la tomamos de los ejemplos de clase para patrones constructores de distintos tipos de objetos, en este caso utilizamos serialización para csv.

Principios aplicados

- **Single Responsibility Principle (SRP)**

Todas las clases implementadas tienen una única responsabilidad como por ejemplo: *'Visual'* que se dedica a toda la interfaz del juego y *'Player'* que cumple la función de tener los datos necesarios para que el jugador pueda jugar.

- **Open Closed Principle (OCP)**

Se crearon clases abstractas como *'Game'*, *'Abstract_Deck'* y *'Abstract_Serializer'* para que se tengan los métodos básicos pero que puedan extenderse agregando nuevas funciones para las clases concretas.

- **Liskov Substitution Principle (LSP)**

Se facilita para el patrón de sustitución de liskov con la clase padre *'Game'*, se puede sustituir una clase hijo en el lugar de la clase padre sin afectar el funcionamiento del juego.

- **Interface Segregation Principle (ISP)**

No aplica para este entregable ya que tenemos una única interfaz posible, los jugadores solo ven el juego.

- **Dependency Inversion Principle (DIP)**

Utilizamos este patrón para depender de abstracciones de decks.