Proyecto de Programación Dominó:

Este proyecto de programación tiene como objetivo fundamental implementar una biblioteca de clases que permita modelar y experimentar con diferentes variantes del juego Dominó, así como diferentes estrategias de jugadores virtuales. Esto fue logrado haciendo uso de las buenas prácticas de desarrollo de software que se han enseñado durante el curso, por lo que se tuvo un especial énfasis en el diseño de las clases, interfaces y métodos al implementarse.

El proyecto fue dividido en dos sub-proyectos:

- *ClassLibrary*
- *GraphicInterface*

ClassLibrary (Biblioteca de Clases): aquí se encuentran todos los componentes principales tales como los objetos, las reglas y la lógica del juego.

GraphicInterface (Interfaz Gráfica): aquí se encuentran todos los componentes gráficos del juego, donde el objeto ***ConsoleInterface*** es el objeto principal de la parte gráfica.

Ahora procederemos a explicar la estructura principal del juego en donde se destacará la función de cada objeto dentro de él.

Primero que todo la estructura fundamental es el objeto ***Tournament*** el cual representa un torneo. Un torneo es un conjunto de juegos sucedidos en un orden topológico, este viene dado por un grafo de dependencias, en donde un juego puede depender de los resultados de otros juegos. Este grafo de dependencias debe ser un grafo acíclico dirigido o DAG (del inglés Directed Acyclic Graph) para que pueda existir un ordenamiento topológico de los juegos. Las aristas que van desde un primer juego hacia un segundo solo indican que los equipos ganadores del primer

juego formaran parte del segundo cuando todas sus dependencias sean completadas.

Los torneos están conformados por objetos ***Match*** los cuales no son más que identificadores que generan un juego con los equipos que formen parte de él. Por otra parte el objeto ***TournamentHistory*** es el encargado de almacenar la historia e información completa del torneo.

Como parte fundamental nos encontramos ahora con el objeto ***Game*** el cual es en si el componente primitivo de cada torneo. Este es el objeto que genera el objeto ***Match*** para que jueguen los equipos y así determinar los ganadores.

Game tiene incorporado varios sub-objetos, los cuales juntos conforman el juego. Entre ellos están:

- ***Box***: es el encargado de generar y almacenar las fichas del juego que luego serán repartidas a cada jugador
- ***Board***: es el tablero de un jugador, en él se almacenan las fichas después de ser repartidas. Cada jugador posee su propio tablero.
- ***Table***: es la mesa, es donde se colocan y se juegan todas las fichas.
- ***History***: es el registro o historial del juego, es donde se almacena la información que puede ser requerida en un futuro.
- ***TeamInfo***: es el encargado de administrar los equipos, los jugadores y el orden de las jugadas

Por otro lado cada juego está distribuido por rondas, que no son más que una secuencia lineal de partidos, donde poco a poco se van actualizando los puntajes de los jugadores y va cambiando la dinámica de las puntuaciones.

Ahora bien, sabemos cómo está conformado el objeto ***Game***, ¿Pero cómo funciona la lógica del juego y sus sucesos?, para ello hacemos uso de ***Events*** y ***States***, que no son más que los componentes primitivos usados en la arquitectura del juego.

El objeto ***Game*** posee varias funciones que modifican y obtienen información de sus sub-objetos, estas funciones son accedidas a través de eventos y estados. Un ***Event*** (evento) es un objeto que modifica los sub-objetos de ***Game***, mientras que un estado es una función que realiza una consulta sobre ellos.

Para lograr darle funcionalidad al juego se crea un grafo de eventos y estados, donde un nodo representa un evento y una arista lleva consigo un estado. Un evento puede ser sencillo o complejo. Un evento sencillo no es más que una acción a ejecutar sobre los sub-objetos de ***Game***, mientras que uno complejo representa un grafo de eventos y estados. Además, un evento complejo tiene un inicio y varios finales, donde el inicio es donde empieza a ejecutarse la secuencia de eventos y los finales es donde al estar en ellos el grafo deja de ejecutar eventos y finaliza su proceso. El orden de los eventos viene dado por el clásico *pre-orden* recursivo en un grafo, sin embargo solo se va desde un nodo a otro si el estado asociado a esa arista retorna verdadero. Un estado, como lo habíamos mencionado anteriormente no es más que una función booleana que realiza una consulta sobre el juego, si la condición se cumple entonces se atraviesa esa arista, en caso contrario se omite esa transición.

Con esta manera de implementar la lógica del juego se pueden generar muchas cosas interesantes, ya que se puede ser tan creativo como se quiera a la hora de mezclar eventos y crear aristas con estados asociados.

Por otro lado para hacer un poco más interesante el juego se implementó el objeto ***Power***, el cual les da cierta capacidad a los jugadores de alterar el comportamiento normal del juego permitiendo saltarse eventos o incluso repetirlos.

Estos ***Powers*** (Poderes) son etiquetas que permiten a los eventos tenerlos en cuenta a la hora de interactuar con el objeto ***Game***. Su principal uso está en las fichas, es decir, que cada ficha puede o no tener varios poderes que son activados una vez es jugada en la mesa.

Para crear un nuevo ***Power*** y que a su vez tenga utilidad se debe trabajar en los eventos para que lo tengan en cuenta en su funcionamiento, por lo que podemos afirmar que los poderes están estrictamente controlados y subordinados al grafo de eventos que es la máxima autoridad en el control de la lógica del juego.

Ahora hablaremos sobre las reglas del juego, que son las que les dan un toque personalizado. Para este proyecto se implementaron varias reglas, entre ellas se encuentran:

- ***IBoxGenerator***: forma de generar las fichas de la caja
- ***IDrawable***: criterio para robar una ficha después de un turno
- ***IGameFinalizable***: criterio de finalización del juego
- ***IGameWinnerRule***: criterio de ganadores de un juego
- ***IIdJoinable***: criterio de compatibilidad de dos caras
- ***IJoinable***: criterio de compatibilidad de dos fichas
- ***IOrderPlayerSequence***: secuencia de orden de los jugadores
- ***IReversePlayerOrder***: criterio de reversibilidad del orden de los turnos
- ***IRoundFinalizationRule***: criterio de finalización de una ronda del juego
- ***IRoundGame***: grafo de eventos y estados que representan una ronda
- ***IRoundScorePlayer***: criterio de puntuación para un jugador en una ronda
- ***IRoundScoreTeam***: criterio de puntuación para un equipo en una ronda
- ***IRoundWinnerRule***: criterio de ganadores de una ronda del juego
- ***IScoreTeam***: criterio de puntuación de un equipo en el juego
- ***ITeamOrder***: manera de distribuir el orden de jugadores por equipo
- ***ITokenDealer***: forma de repartir fichas a los jugadores al principio
- ***ITokenValue***: forma de calcular el valor de una ficha
- ***ITokenVisibility***: visibilidad de una ficha con respecto a los demás jugadores
- ***ITournamentGenerator***: forma de generar el torneo
- ***ITournamentWinnerRule***: criterio de ganadores en un torneo