1 Diseño y evolución de las clases principales del Modelo

1.1 Diseño del tablero

Toda la lógica que implementa el concepto del tablero queda reflejada a lo largo de todo el proyecto en la clase Board.

1.1.1 Sprint 1

La clase Board es un simple contenedor de cubos, organizados en forma de matriz. No tiene interacción con otros objetos del modelo.

1.1.2 Sprint 2

Ahora la clase Board tiene la funcionalidad de actualizarse a sí mismo una vez se coloca un nuevo cubo, funcionalidad que antes estaba delegada a la clase Game.

1.1.3 Sprint 3

A partir de este Sprint los tableros tienen forma (es decir, la forma no es necesariamente siempre cuadrada) y tamaño elegido por los usuarios. A parte, Board tiene una representación en forma de String y de JSONObject, a través de los métodos toString() y report().

1.1.4 Sprint 5

Ahora, aparte de guardarse los cubos del tablero en forma de matriz, se guardan en forma de lista por ser una representación de los datos muy conveniente en distintas partes del proyecto, entre otras, para la red.

1.2 Diseño de los colores

1.3 Diseño de los cubos

1.4 Diseño del juego

La clase fundamental del juego es la clase Game, la clase principal del modelo.

1.4.1 Sprint 1

La clase Game responde a ejecuciones del PlaceCubeCommand, colocando nuevos cubos en las casillas seleccionadas, actualizando el tablero, manejando los turnos de los jugadores y actualizando sus puntos.

1.4.2 Sprint 2

A partir de este momento, Game no se encarga de actualizar el tablero y las puntuaciones de los jugadores, sino de únicamente pasarle al tablero los cubos nuevos que tiene que insertar. El tablero, a partir de ese cubo, se actualiza a sí mismo, y los cambios de cubos actualizan las puntuaciones de los jugadores.

1.4.3 Sprint 3

Game tiene su propia representación en forma de String y de JSONObjet, a través de los métodos toString() y report().

1.4.4 Sprint 4

A partir de este sprint, Game pasa a ser una clase abstracta y se pasa a tener dos modos de juego, el clásico (jugadores individuales) y el modo por equipos, que son implementados por las clases herederas de Game: GameClassic y GameTeams. También se implementa el patrón MVC, por lo que Game (modelo) pasa a tener una lista de observadores y métodos para el envío de notificaciones a estos.

1.4.5 Sprint 5

Game se adapta con ciertos cambios y nuevos métodos para soportar el juego en línea.

1.4.6 Sprint 6

En este momento Game sufre su mayor refactorización. Esta clase pasa a extender de la clase Thread, de forma que ya no funciona ejecutando comandos en el mismo momento de su creación, sino que estos de ponen en espera y Game, que está en todo momento funcionando y comprobando si hay nuevas peticiones puestas en espera, las ejecuta cuando puede. Esto nos permite evitar problemas de desbordamiento con el cálculo de jugadas por parte de las inteligencias artificiales, aparte de permitir el funcionamiento esperado de la vista sin comprometer su rendimiento. Aparte, Game deja de llevar a cabo el manejo de turnos y se delega esa responsabilidad a la clase TurnManager, que es invocada tras cada jugada para que ejecute (si procede) el siguiente turno.

1.5 Diseño de los jugadores

1.6 Diseño de los equipos

1.7 Diseño del gestor de turnos

1.8 Diseño de los estados del juego

1.9 Diseño de las replays

1.10 Diseño de las Inteligencias Artificiales

1.10.1 Sprint 5

Las estrategias son externas al modelo. El cómputo de movimientos a través de estas estrategias de hace a partir de la vista, a través de la clase PlayerView, que representa al Player en la vista y se encarga de ejecutar sus acciones. Por razones de encapsulación, las estrategias no tienen acceso al modelo y realizan sus cálculos a través de GameStates.

1.10.2 Sprint 6

Las estrategias ahora forman parte del modelo, siendo atributo de aquellos Players controlados por la máquina. Como en este punto el manejo de turnos se lleva a cabo por la clase TurnManager y el modelo funciona en su propia hebra, la clase PlayerView desaparece del proyecto y son los propios players quienes ejecutan las estrategias, que a nivel abstracto resulta mucho más intuitivo. Por la hebra del modelo, cuando las estrategias terminan de calcular el siguiente movimiento, este no se ejecuta en ese mismo instante, sino que se deja en espera en el modelo hasta que este pueda ejecutarlo.