REFACTORIZACIÓN DEL SPRINT 1

Grupo PMC

1 de abril de 2022

SaveLoadManager

Esta clase pasa a ser exclusivamente un intermediario entre la clase *Game* y los ficheros de carga y guardado y sus métodos deben ser estáticos para que puedan usarse más adelante como componente aislado y no sea necesario instanciarse. Debe tener todo lo necesario para encapsular dicha funcionalidad, entre las características de la clase:

- Debe tener funciones de carga y descarga que llamamos:

```
public static void loadGame();
public static void saveGame();
```

- Recibe una lista de Players, el turno actual y el tablero a través de una interfaz *Saveable*.
- Posee los ficheros de carga y guardado.

Cube

Esta clase queda prácticamente igual a la de la versión anterior, a excepción de que en vez de tener una instancia de la las *Color*, ahora pasa a tener una instancia del *Player* al que pertenece. Gracias a esta modificación reúne las siguientes funcionalidades:

- Modifica puntuaciones del jugador que tiene como instancia (al cambiarse de color cuando juegas, él gestiona el cambio de puntuaciones individualmente). Para ello debe tener una función que dado un color, se autoasigne el jugador de dicho color:

```
public void changeOwner(Color color){
        this.player = ??? // Funcion que busque un jugador en base a un player
}
```

- Tiene acceso al color del player, ya que el tablero solo trabaja con player y debe preguntarle a los cubos su color.

Board

El tablero ahora reúne toda la funcionalidad que es la parte física del juego. Representa íntegramente el tablero y las bolas tal y como sería en el juego físico, por tanto, reúne toda la funcionalidad de poner bolas, cambiarlas de color y gestionar lo relativo a lo que cambia en el tablero físico.

En consecuencia, reúne las siguientes funcionalidades:

- Añade y quita cubos
- Tiene tamaño propio
- Cualquier modificación que se haga sobre un cubo se debe pasar por board
- NO trabaja con *Players*, trabaja con *COLORES*.

- Para cambiar un cubo de color, se delega la responsabilidad en el cubo que sabrá dado un color a que jugador cambiarse y devolverá el color correspondiente a dicho jugador.
- Posee una función que dada un cubo, lo coloca y cambia el tablero completamente:

```
public void update(){
// Actualizamos los cubos y el tablero
}
```

Player

El player de ahora pasa a tener más peso al suprimir la visión de colores al resto de clases que manejan el juego y que no son *Board*. Ahora, para que los cubos puedan cambiarse de poseedor, tienen todos un atributo estático en el que son capaces de consultar, dado un color, el jugador que corresponde y así poder devolver a los cubos lo que necesita:

```
public class Player(){
        private static Player[] players = new Player[Color.size()];
        private Color color;
        private int score;
        private String name;

        ...

        public Player getPlayer(Color color){
        return this.players[color.ordinal()];
        }
}
```

Game

La clase Game a perdido peso en cuanto a lo que la lógica de poner una bola respecta, puesto que se apoya más en la clase Board que gestiona toda la modificación de cubos. Sin embargo, no queda como una clase inútil, sino que se encarga de gestionar la parte externa del desarrollo de una partida. Del mismo modo que el tablero era la parte física del juego, esta clase puede considerarse como el "árbitro" que va dando orden a los jugadores, prepara para guardar y cargar el juego y coordina al resto de elementos del juego para colaborar entre sí. Reune las siguientes funcionalidades:

- Tiene que poder preparar un formato adecuado del estado del juego para que la clase *SaveLoadManager* guarde el juego correctamente.
- Tiene que tener una función que juegue un turno completo y que cambie de jugador al siguiente al final.
- El método update que tenía ahora pasa al tablero y él solo lo llama con la información adecuada durante el turno correspondiente.
- En su constructor solo recibe un *Board*.
- Es creado en *Controller*, aunque pretendemos mejorar esto en futuros sprint.

Controller

La clase Controller está todavía un poco a expensas de que se desarrollen otras clases que están en proceso como los commandos. Sin embargo, por lo pronto sabemos que tiene que haber:

- Un menú que permita elegir entre los diferentes commandos

- Un método para pedir los nombres y el número de jugadores en caso de querer jugar y no cargar ni guardar partida.
- Tiene que haber un método que se dedique a hacer correr la aplicación:

```
public void run();
```