**TRABAJO PRÁCTICO**

***Programación Orientada a Objetos***

**Integrantes**

* **Besteiro, María Florencia** *<51.117>*
* **Rivas, Leandro Matías** *<51.274>*

**2012**

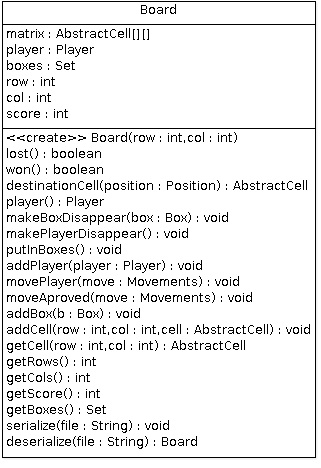
**1º Cuatrimestre**

**Jerarquía de Clases**

Nuestro backEnd consta de 3 clases principales:

* Board
* AbstractCell
* MovableObject

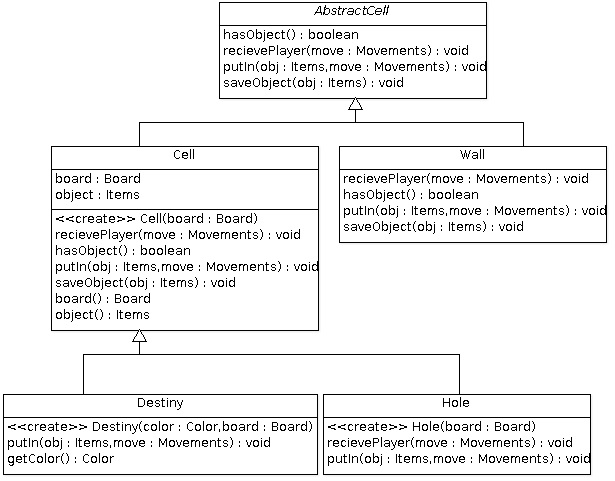
**Board**



Esta clase representa el tablero del juego que contiene piezas de distinta índole: paredes, destinos, cajas, agujeros, etc. Esta modelada como una matriz de objetos de tipo **AbstractCell** la que se detalla más adelante. Esta clase Board, aparte de contener una matriz, contiene un objeto de tipo Player, una colección (Set) de cajas y un puntaje.

Luego de crear el tablero la clase Parser crea las celdas y las guarda en sus posiciones correspondientes con el método *addCell*.Por otra parte, las cajas son acumuladas en un Set mediante el método *addBox*, para luego agregarlas a las celdas correspondientes (las celdas tienen una referencia a las cajas que se guarda en la variable “object” que poseen). El método que se encarga de esto último es *putInBoxes*.

**AbsractCell**

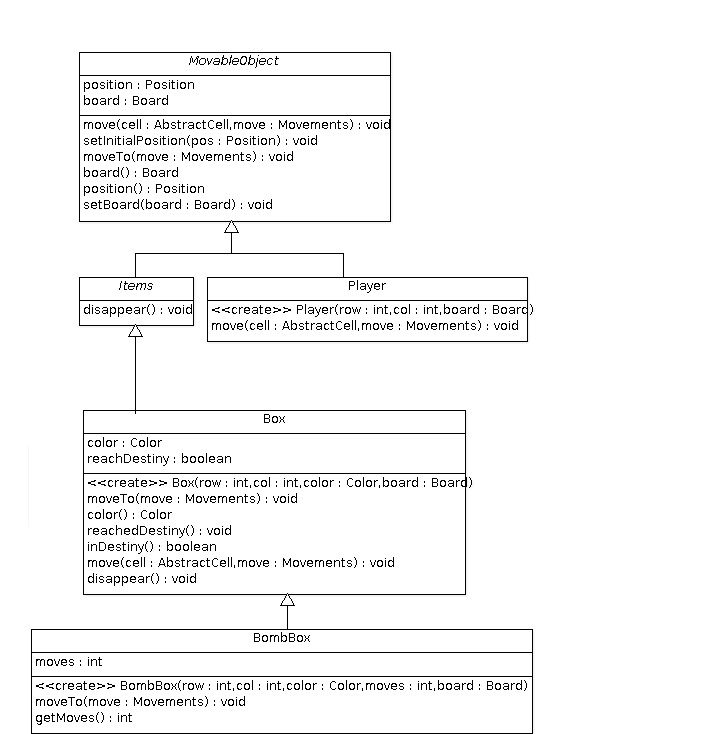


Esta clase modela el comportamiento común que poseen todas las piezas NO movibles dentro del tablero. Esta clase fue creada para hacer uso de polimorfismo y así poder hacer que el tablero tuviera una matriz con celdas que respondieran a las mismas responsabilidades:

* Responder si posee un objeto ( *hasObject* )
* Recibir al jugador ( *recievePlayer* )
* Poner un objeto ( *putIn* )
* Guardar el objeto ( *saveObject* )

Al ser el modelo de comportamiento, sus métodos son abstractos (los arriba mencionados) lo que genera que toda subclase de AbstractCell tiene que sobrescribir estos métodos, ya que no todas las piezas se comportan de la misma manera al recibir un jugador, poner un objeto, etc. Además, se ha busco que puedan recibir elementos del tipo Items, es decir, cajas (en este caso) o cualquier otro ítem que queramos agregar al juego dado que estas subclases sabrán como interactuar con ellos.

**MovableObject**



Esta clase abstracta modela el comportamiento de los objetos MOVIBLES del juego. Ya que no todos los objetos movibles se mueven y comportan de la misma manera. En nuestro primer diseño nos encontramos con el problema de la poca extensibilidad del código, por lo que creamos esta clase y la clase *Items*  que por el momento solo posee el método abstracto disappear() pero fue creada para la extensibilidad en el caso que se deseen crear nuevas piezas movibles en el juego.

**Subclases de AbstractCell**

**Cell**

Modela una celda vacía. Esta puede almacenar un objeto de tipo Item. Si no posee ningún elemento el jugador puede moverse, si no, intenta mover el objeto llamando a su método *move* (método que poseen los objetos de tipo MovableObject) si este se movió, el jugador se mueve.

**Destiny**

Subclase de *Cell.* Utiliza los métodos de su clase padre sobrescribiendo solamente *putIn* ya que cuando recibe una caja, si esta es del mismo color que el destino, llama al método de la misma *reachedDestiny* comunicándole que llegó a su destino correspondiente.

**Hole**

El comportamiento de esta clase consiste en desaparecer cualquier objeto que reciba. En el caso de recibir un jugador, lo hace desaparecer y se pierde el juego. En el caso de desaparecer una caja puede continuarse jugando pero la caja se pierde sin poder recuperarse.

**Wall**

Esta celda funciona como su nombre lo dice, una pared. No permite moverse por ella a ningún objeto de tipo MovableObject.

**Subclases de MovableObject**

**Player**

Modela a un jugador que conoce su posición y el tablero. Cuando puede moverse lo hace seteando su posición hacia una nueva según la dirección en la que se mueve.

**Box**

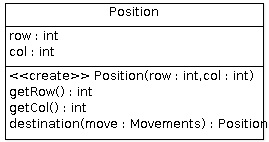
Esta clase posee un color, su posición y una referencia al tablero. A su vez posee un flag que devuelve boolean si llegó a su destino o no. Cada vez que se mueve, cambia este flag a *false* ya que sale de su destino. Este objeto como subclase de MovableObject puede moverse y cuando lo hace llama al mensaje *moveAproved* de board que hace que el jugador se pueda mover.

**BombBox**

Subclase de Box. El único cambio en su comportamiento, respecto del de su clase padre Box es que posee un contador llamado *moves* que indica al inicio, la cantidad de movimientos que tiene permitidos antes de explotar. Cada vez que este objeto se mueve su contador baja y cuando llega a 0 explota, haciendo que el jugador desaparezca con el método *makePlayerDisappear* que posee board.

**Clases Auxiliares**

**Position**

****

Es una clase modelada para facilitar las posiciones de las celdas, jugador, etc. Posee un método llamado *destination* que recibe un movimiento y a partir de la posición en si y el movimiento genera una posición nueva.

**Cambios de diseño**

En un principio al diseñar el juego, nuestro diseño fue distinto. Las clases de MovableObject e Item no existían. Player y Box no heredaban de ninguna clase. La celda abstracta contaba con métodos:

* receivePlayer
* receiveBox
* putInBox
* withBox

Al encontrarnos con diferentes problemas, uno de ellos fue la poca extensibilidad del diseño ya que los métodos fueron creados particularmente para los objetos del juego y el querer agregar cosas nuevas como barriles, por ejemplo, tendría que cambiar el código significativamente. Fue por esto que volvimos a la idea principal que tuvimos sobre el diseño de clases y crear una clase que representara a los objetos movibles del juego y de esa manera llegar a al diseño final que se detalló anteriormente.