FIUBA - 7507

Algoritmos y programación 3

Trabajo práctico 2: BombitaRodriguez

1er cuatrimestre, 2012

(trabajo grupal)

Nombre	Padrón	E- mail
Leandro Tullio	93214	letullio@gmail.com
Marcelo Cavazzoli	93429	chelo_c@live.com
Mateo Bosco	93488	mateo.bosco@hotmail.com
Juan Manuel Hidalgo	93383	juanmah_@hotmail.com

Página en blanco (intencionalmente)

Supuestos

Suponemos que las “ciudades” no deben estar generadas aleatoriamente sino que están diseñadas por nosotros.

Modelo de dominio

Comenzamos el trabajo pensando las clases necesarias y sus relaciones para luego volcarlo en un diagrama de clases. Para empezar pensamos los casos de uso más específicos y escribimos las pruebas que evalúen esos casos. Luego escribimos el código de las clases para que pasen las pruebas. Una vez hechos los casos de usos más específicos, fuimos haciendo pruebas de integración de las distintas clases para validar su correcto comportamiento y la interacción entre ambas. Terminado el modelo hicimos el controlador del teclado y la interfaz gráfica.

Diagramas de clases

El bomberman (juego en general) es controlado principalmente por la clase *Juego*, la cual se encarga de relacionar las clases que interactúan entre ellas. Esta clase contiene a todos los objetos que se encuentran activos en el juego, en las variables: *ArmamentosVivos, EnemigosVivos, Obstaculosvivos, ArticulosVivos*. También contiene al jugador principal al cual lo hace interactuar con los objetos. Además la clase Juego, es encargada de validar los movimientos y las acciones. (por ej: poner una bomba).

La clase *ArmamentosVivos* contiene a las instancias de la clase *Armamento* que son las bombas próximas a explotar.

EnemigosVivos contiene a las instancias de la clase *Enemigo* que puede ser del tipo: *Cecilio, Lopez Reggae o Lopez Reggae Alado*.

ObstaculosVivos contiene a las paredes del “nivel”, las instancias de la clase *Obstaculo* que puede ser *BloqueAcero, BloqueLadrillo o BloqueCemento.*

Por último la clase *ArtículosVivos* contiene las instancias de la clase *Artículos* que puede ser: *Chala, BuffToleTole o Timer.*

A su vez, la clase *Juego* contiene una instancia de la clase *Puerta* que sirve para pasar de nivel.

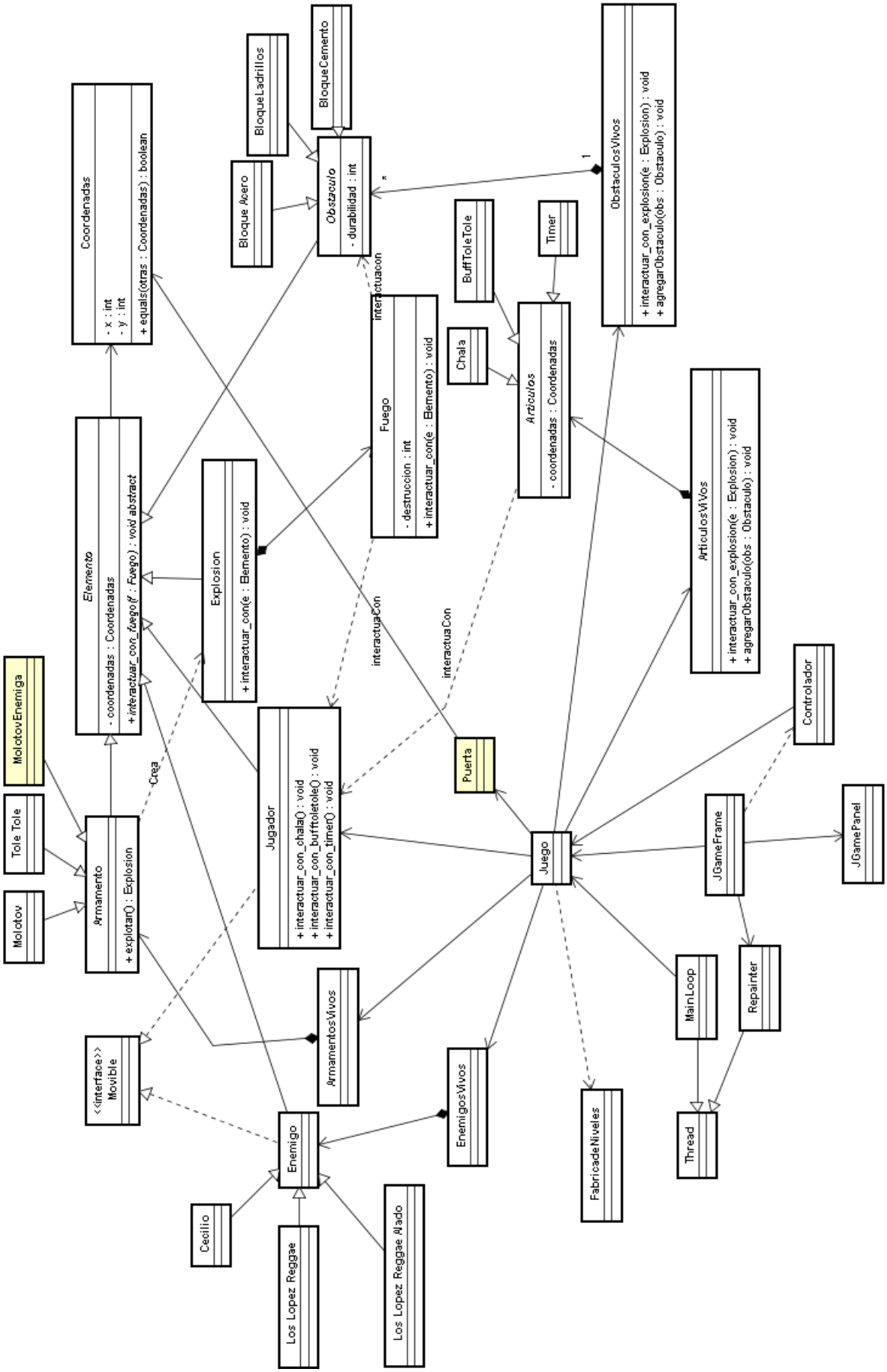
Diseñamos la clase Explosión la cual consiste en una colección de instancias de la clase *Fuego*, los cuales son los que al interactuar con los jugador, enemigos y paredes les quita vida.

Luego tenemos un hilo (Thread), Mainloop, el cual es realiza ciclos infinitos llamando a ciertos métodos de la clase Juego.

Para la interfaz gráfica usamos el patrón MVC, separando la vista del modelo, usando la librería Swing (javax.swing) creamos una ventana y otro hilo (Thread) Repainter, que es el encargado de repintar la pantalla.

Siguiendo con la estrategia MVC separamos el controlador del juego del modelo y la vista, en donde tenemos los observadores y son los encargados de avisar al modelo sobre alguna acción que realiza el usuario.

Por ultimo usamos la librería DOM4J para incorporar persistencia y serializarían en nuestro proyecto. Cada clase sabe cómo serializarse y como recuperarse de un archivo XML.


Coordenadas
- x : int
- y : int
+ equals(otras : Coordenadas) : boolean
BloqueLadrillos
BloqueCemento
Bloque Acero
Obstaculo
- durabilidad : int
ObstaculosVivos
+ interactuar_con_explosion(e : Explosion) : void
+ agregarObstaculo(obs : Obstaculo) : void
1
*
interactuacon
Fuego
- destruccion : int
+ interactuar_con(e : Elemento) : void
BuffToleTole
Timer
Chala
Articulos
- coordenadas : Coordenadas
ArticulosVivos
+ interactuar_con_explosion(e : Explosion) : void
+ agregarObstaculo(obs : Obstaculo) : void
Elemento
- coordenadas : Coordenadas
+ interactuar_con_fuego(f : Fuego) : void abstract
Explosion
+ interactuar_con(e : Elemento) : void
interactuaCon
interactuaCon
MolotovEnemiga
Tole Tole
Molotov
Armamento
+ explotar() : Explosion
Crea
Jugador
+ interactuar_con_chala() : void
+ interactuar_con_bufftoletole() : void
+ interactuar_con_timer() : void
Puerta
Juego
Controlador
JGameFrame
JGamePanel
MainLoop
Repainter
Thread
<<interface>>
Movible
ArmamentosVivos
EnemigosVivos
Enemigo
FabricadeNiveles
Cecilio
Los Lopez Reggae
Los Lopez Reggae Wado

Detalles de implementación

En este TP pudimos aplicar todos los conceptos aprendidos de polimorfismo y herencia como sobrecarga de métodos, interfaces, patrones de diseño, persistencia, etc.

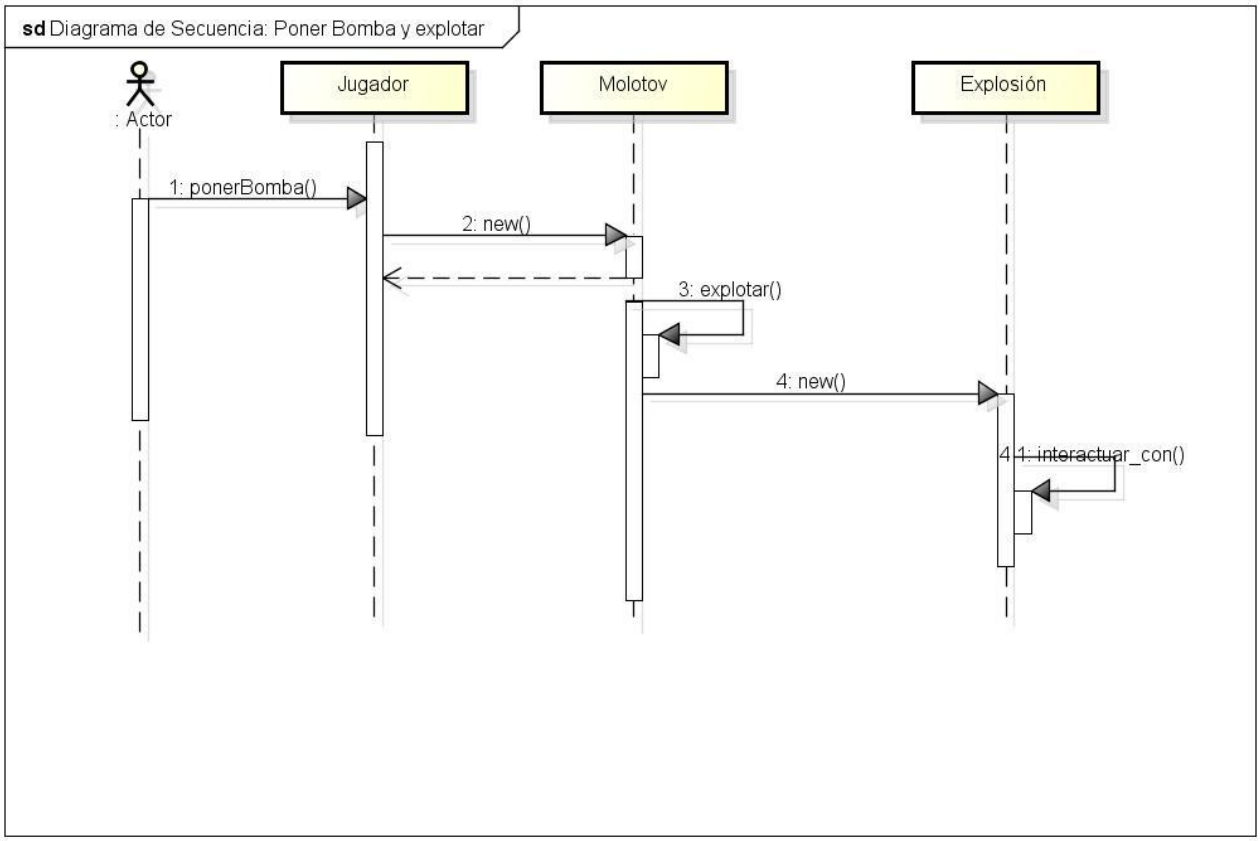
Patrones utilizados: Factory, Double Dispatchment, Iterator, MVC.

La estrategia utilizada fue pensar los casos más específicos en el cual describimos las clases y sus métodos repitiendo esto con todos los casos que se nos ocurrieron. Luego fuimos agregándole complejidad buscando casos de uso en donde interactúen varias clases y haciendo pruebas para cada interacción. Cada vez que se realizaba un cambio se fueron corriendo todas las pruebas para ir verificando que se pasen todas las pruebas, las viejas y las más nuevas. Fuimos haciendo todos los métodos y clases descriptos para corroborar que pasen correctamente las pruebas. Este es un claro uso de Test Driven Development.
Para solucionar el problema de las interacciones entre las paredes, los jugadores y las bombas utilizamos el patrón Double-Dispatch.
Para los objetos que se pueden mover implementamos una interfaz llamada Movibles para que puedan utilizar el método *mover* dependiendo de cómo se puedan mover en el juego.

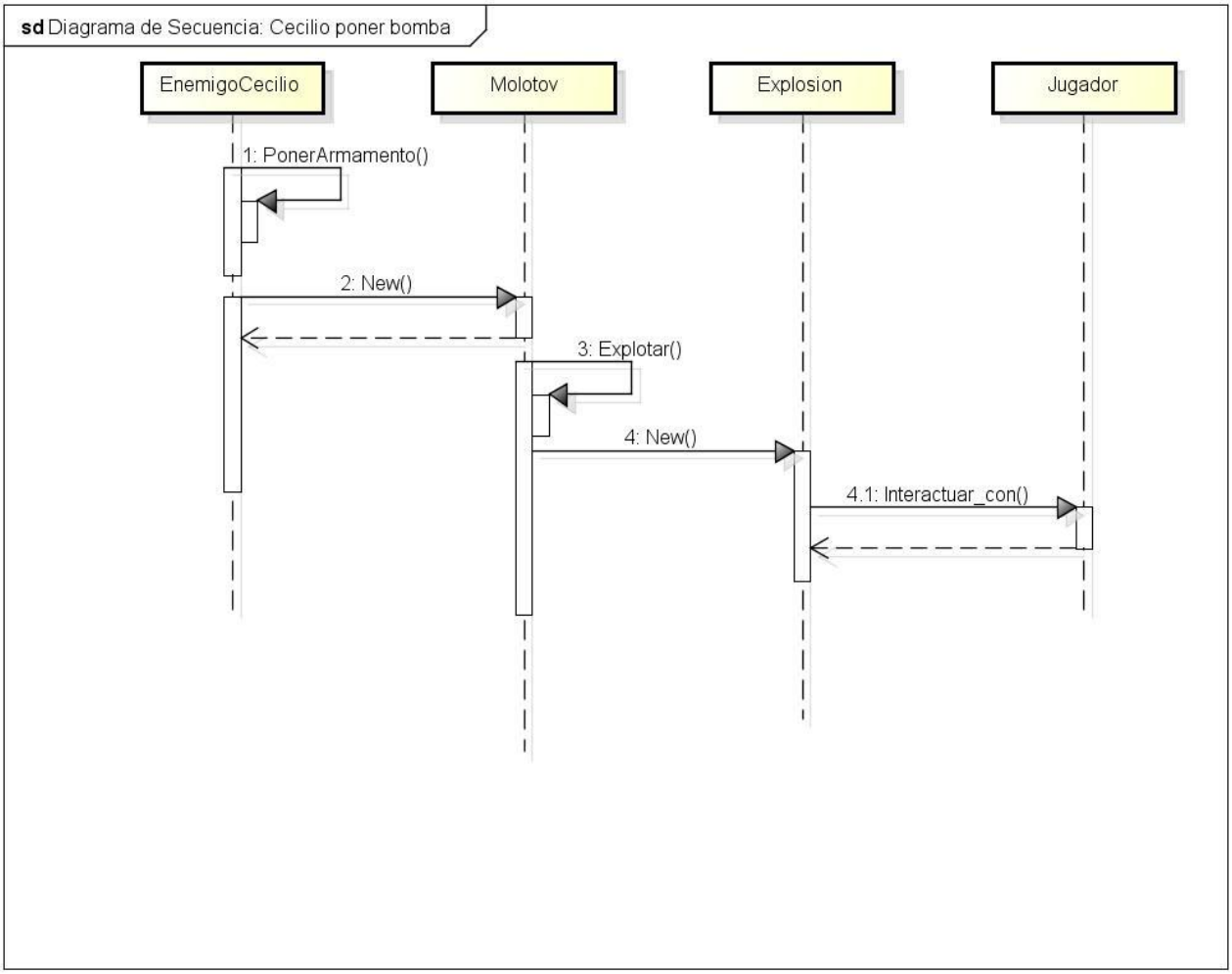
Diagramas de secuencia

Nota: No hemos utilizado al máximo los diagramas de secuencia ya que nos basamos mucho en TDD, en donde prácticamente hay diagramas de secuencia en código. Sin embargo a continuación se muestran ejemplos.

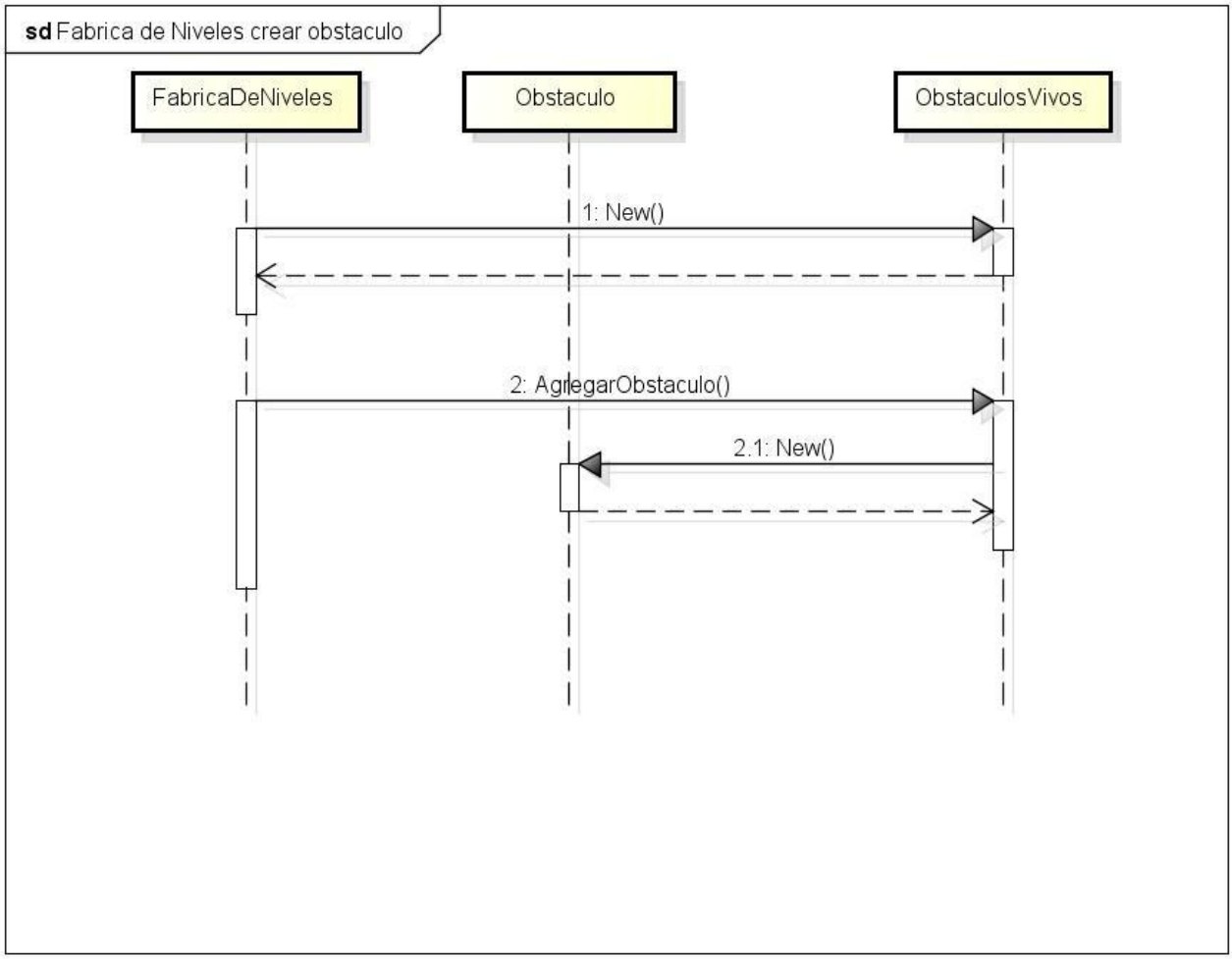
Jugador poner bomba y explotar.



Enemigo Cecilio poner bomba.



Fabrica de Niveles crear los obstáculos Vivos.



Elemento Interactuar con Explosion (Double dispatchment).