**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: AlgoCraft*

1er cuatrimestre, 2015

(trabajo grupal)

Alumnos:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **PuertoEstelarTerranMail** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

***Fecha de entrega final***: Miércoles 24/06/2015 - Jueves 25/06/2015

***Tutor***:

***Nota Final***:

# 

Informe

Supuestos

A la hora de realizar el Trabajo Práctico, se establecieron ciertos supuestos respecto a las reglas del juego AlgoCraft. En principio, se determinó que cada jugador tendría asociada una única Base, la cual se encargaría de administrar sus recursos. A su vez, para que uno de ellos quede eliminado de la partida, su base tendría que ser destruida. También se fijó que cuando a un jugador le destruyen algún centro recolector de suministros, ya sea un Centro de Mineral o una Refinería, el mismo no perderá los recursos que hayan sido recolectados por dicho edificio. Otra idea implementada fue que todas las unidades tuvieran el mismo rango de movimiento y que cada jugador tuviese un rango alrededor de la base, dentro del cual puede construir sus edificios; dicho rango irá aumentando por cada edificio que construya.

## Modelo de dominio

Diagramas de clases

Se encuentran adjuntos al Informe.

Diagramas de secuencia

Se encuentran adjuntos al Informe.

Diagrama de paquetes

Se encuentran adjuntos al Informe.

Diagramas de estado

Se encuentran adjuntos al Informe.

Detalles de implementación

Un detalle de implementación importante es que la clase Mapa fue implementada utilizando el patrón de diseño Singleton ya que todas las entidades del programa necesitan acceder al mismo (por ejemplo, cuando se crea una unidad, la misma es agregada al mapa en el Constructor por lo que se necesita tener una referencia al mapa). Se implementaron los métodos reiniciarInstanciaParaTest y limpiarMapa con el objetivo de poder borrar los elementos del mismo; el primero es invocado al comienzo de los tests que utilizan el mapa para poder garantizar que el mismo se encuentra vacío, mientras que el segundo se lo usa cuando se desea comenzar una nueva partida.

Otro detalle para destacar es la utilización del patrón de diseño Abstract Factory en la clase Jugador, para poder crear los disntintos edificios y unidades sin repetir código. Fue necesario crear dos Abtracts Factories, una para los edificios, llamada y otra para las unidades (ya que la forma de crearlas dentro de la clase Jugador es diferente), las cuales se llaman CreadorDeEdificios y CreadorDeUnidades respectivamente; de estas clases heredan las Factories que crean las diferentes unidades y edificios (como se puede observar en los diagramas de clase).

A la hora de programar la interfaz gráfica, se creó una clase madre vista de la cual heredan todas las demás vistas, habiendo una para cada clase de objeto que tenga que aparecer en la interfaz.

Excepciones

# 

# **Checklist de corrección**

Esta sección es para uso exclusivo de los docentes, por favor no modificar.

## Carpeta

**Generalidades**

* ¿Son correctos los supuestos y extensiones?
* ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

**Modelo**

* ¿Está completo?¿Contempla la totalidad del problema?
* ¿Respeta encapsulamiento?
* ¿Hace un buen uso de excepciones?
* ¿Utiliza polimorfismo en las situaciones esperadas?

## Diagramas

**Diagrama de clases**

* ¿Está completo?
* ¿Está bien utilizada la notación?

**Diagramas de secuencia**

* ¿Está completo?
* ¿Es consistente con el diagrama de clases?
* ¿Está bien utilizada la notación?

**Diagrama de estados**

* ¿Está completo?
* ¿Está bien utilizada la notación?

# **Código**

**Generalidades**

* ¿Respeta estándares de codificación?
* ¿Está correctamente documentado?