FACULTAD DE INGENIERIA
Universidad de Buenos Aires

75.10 Técnicas de Diseño
2° Cuatrimestre 2014

Corrector: Carlos Curotto

GRUPO 7

Integrantes:

Bosco, Mateo	93488	mateo.bosco@hotmail.com
Devoto, Gabriel Alejandro	89669	gabrielalejandrodevoto@gmail.com
Solotun, Roberto	85557	rsolotun@gmail.com

1 Tabla de Contenidos

2 Introducción

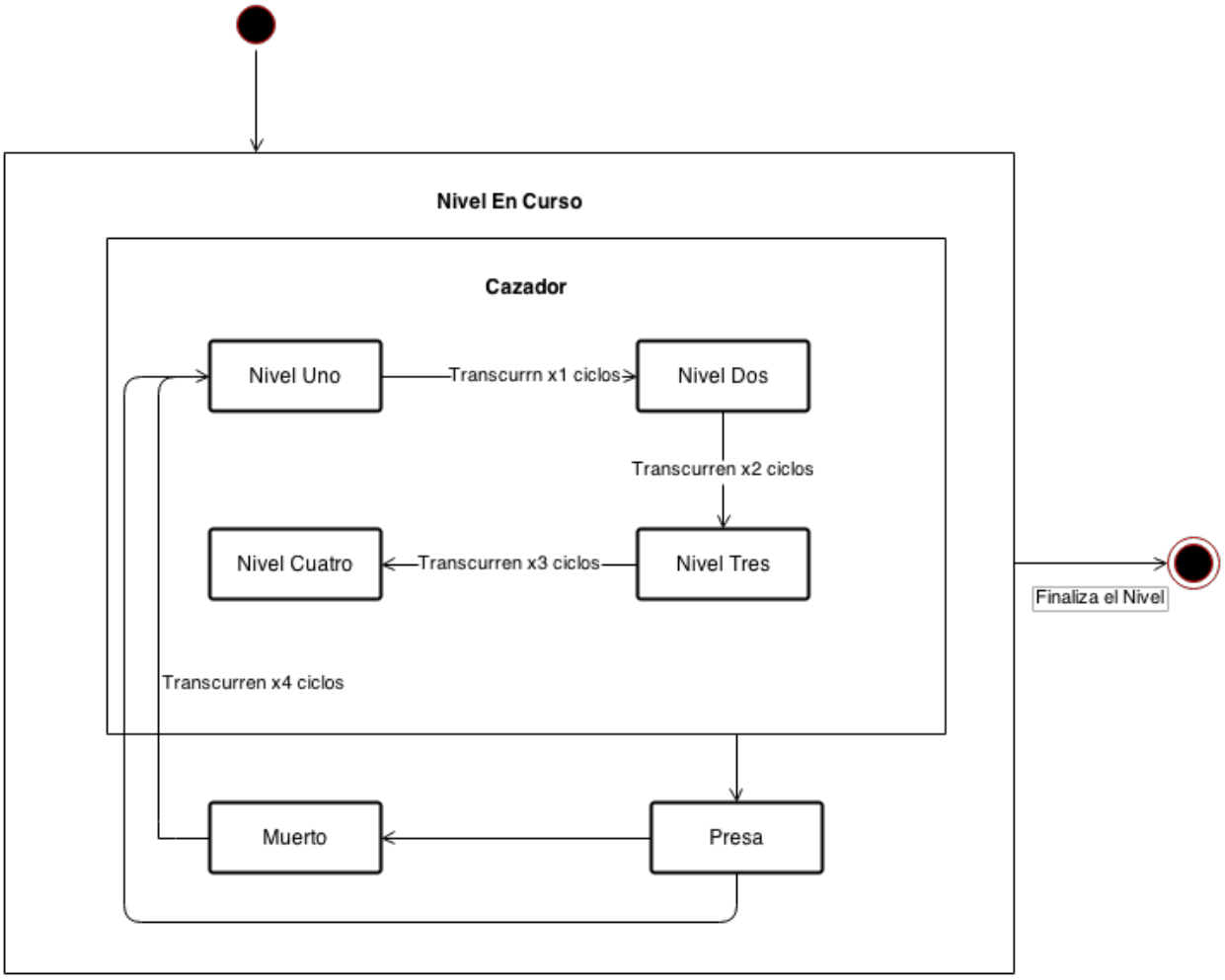
El siguiente informe corresponde a la documentación del Trabajo Práctico N° 1 para la implementación de un PacMan.

El objetivo de esta iteración es la de modelar el dominio descrito en el enunciado siguiendo las buenas prácticas vistas en clase y escribir un buen set de pruebas que se apegue al enunciado para terminar de entender el modelo y que el diseño sea el mejor para cumplir con lo pedido.

Para cumplir con el objetivo se utilizarán las herramientas **JUnit, Maven, Repositorio Git**.

3 Diseño, Implementación y Consideraciones Generales

Para el diseño del modelo primero comenzamos haciendo un diagrama de estados con todos los cambios de estados o de niveles que podía tener el fantasma y luego de refinarlo leyendo varias veces el enunciado llegamos a este diagrama:



Luego fuimos definiendo un set de pruebas en donde se testean cada uno de los escenarios posibles en cuanto a cambios de estados del fantasma, cambios de posición, cambios de nivel de ira, etc y además un set de tests unitarios para las clases que implementamos siguiendo las buenas prácticas vistas en clase.

Al abstraernos de la implementación del código, pudimos definir bien las reglas del modelo por ejemplo haciendo un test por cada cambio de estado permitido y cada cambio de estado no permitido que tenía el fantasma

Cambio permitido:

```
public void fantasmaPresaSeMuere() {
    Fantasma unFantasma = new Fantasma(new Posicion(numeroAleatorio(),numeroAleatorio()));

    unFantasma.convertirAPresa();
    unFantasma.morir();

    assertEquals(EstadosFantasma.MUERTO, unFantasma.estado() );
  }
```

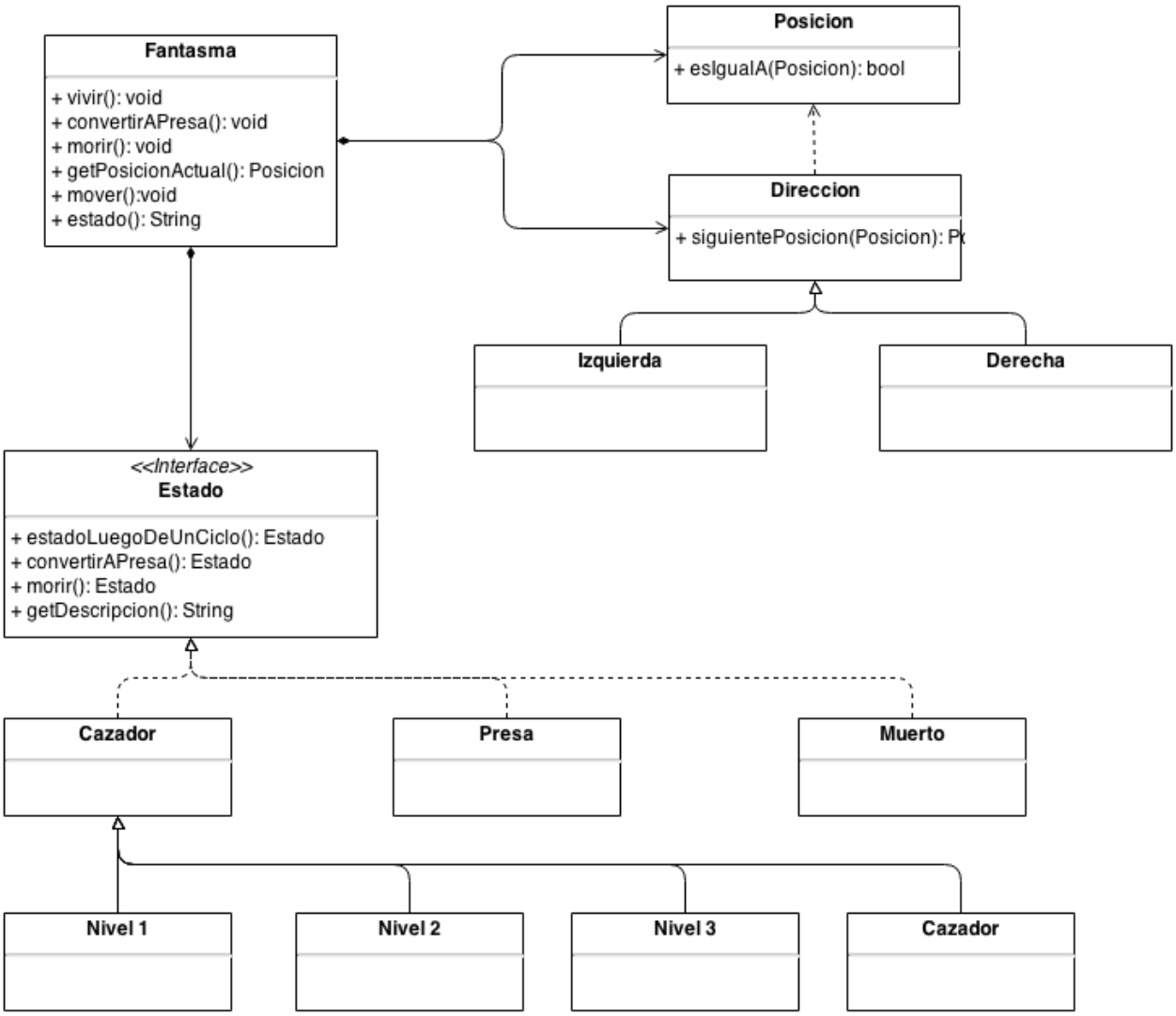
Cambio no permitido:

```
public void fantasmaCazadorNivelUnoNoMuere() {
    Fantasma unFantasma = new Fantasma(new Posicion(numeroAleatorio(),numeroAleatorio()));

    unFantasma.morir();

    assertEquals(EstadosFantasma.CAZADOR_NIVEL_UNO, unFantasma.estado() );
  }
```

Luego de definir el set de pruebas, comenzamos con el diseño del modelo. Este es el diagrama de clases:



4 Conclusiones

Al comienzo se crearon dudas en cuanto al enunciado ya que el mismo no plantea requerimientos en cuanto al comportamiento del fantasma (parte del juego del Pacman sobre lo que se enfoca esta iteración). Más bien deja entrever una posible solución a lo que podría ser un problema de cambio de comportamiento por parte del fantasma frente al paso del tiempo o como consecuencia de futuras interacciones con otros personajes del videojuego.

Por ejemplo, el enunciado plantea estados del fantasma que se suceden unos a otros en consecuencia del paso del tiempo.

Cazador Nivel 1 -----> Cazador Nivel 2 -----> Cazador Nivel 3 .....

Sin embargo, no se especificaba un cambio de comportamiento en el mismo. Los estados simplemente se diferenciaban por el nombre y no por tener un comportamiento real distinto uno del otro. Esto dificultó, en principio, la implementación de pruebas sobre el fantasma, no se sabía que esperar (Assert) luego del estímulo que generaba un cambio de estado (Act).

La solución propuesta fue crear métodos de acceso al estado del fantasma y que estos pudieran darnos una descripción de sí mismos para corroborar que cambiaban de acuerdo a lo que se pedía en el enunciado. Somos conscientes de que esto claramente viola el principio de ocultamiento de la información, ya que estamos revelando detalles de implementación sobre de cómo el fantasma cambia su comportamiento (o mas bien, futuro comportamiento). En lo posible esto será enmendado en las próximas iteraciones.

Con respecto al requerimiento de que el fantasma se pueda mover fue más directo y sencillo. Las pruebas atacan directamente al problema y los detalles de la implementación de la solución nunca son revelados hacia el usuario.

La susodicha solución se basa en que el fantasma posee una posición actual y se mueve en una dirección. Para obtener el próximo punto en el cual se encontrará, simplemente se pide a la dirección el siguiente punto en esa dirección. La abstracción creada sobre la dirección nos permitirá más adelante crear nuevas direcciones sin necesidad de que las clases que las usan se enteren. De

hecho, para esta iteración sólo fueron creadas 2 direcciones concretas, suficientes para validar el requerimiento de movimiento en el enunciado.