FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

75.10 – TÉCNICAS DE DISEÑO

TRABAJO PRÁCTICO PACMAN

Primera Iteración

2do cuatrimestre 2014

Grupo 1

93083 – Fernández, Daniel
91462 – Rojas, Agustin
91958 – Sella Faena, Jasmina

Indice

1. Objetivo

El objetivo de la primera iteración es diseñar e implementar un fantasma del juego, teniendo en cuenta los distintos estados por los que transita a medida que pasa el tiempo y cómo cambia su comportamiento según el estado.

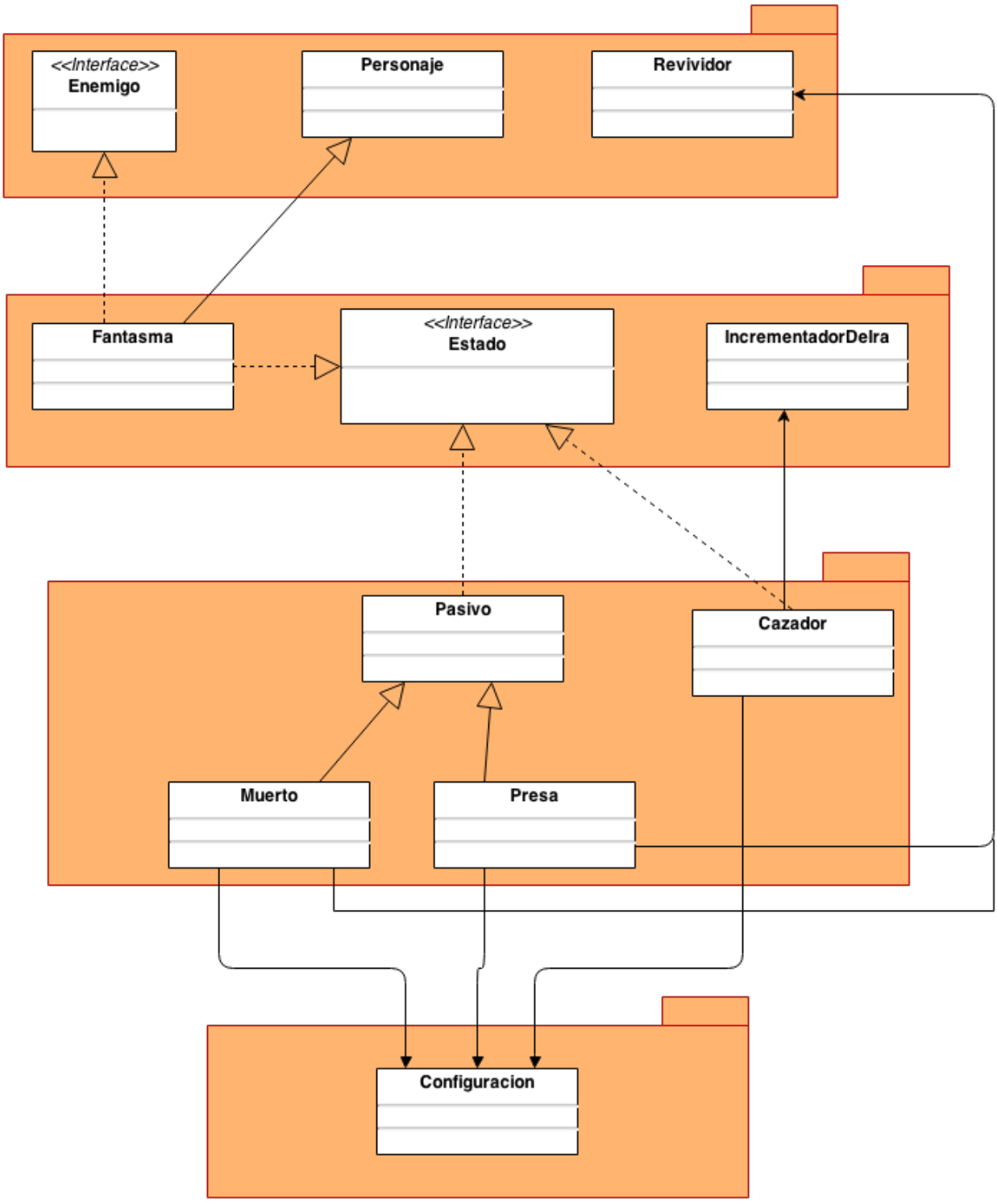
2. Decisiones para la resolución del problema

Para el manejo de las transiciones entre estados se decidió que los distintos estados posibles implementen una interfaz común que modele las acciones que los puede afectar y/o modificar el estado del fantasma. El tiempo involucrado en estas transiciones se cuenta desde las clases Cazador, Presa y Muerto, las cuales serán las encargadas de llamar a los métodos de la clase Fantasma encargados de los cambios de estado correspondientes, evitando de esta forma que la clase Fantasma se pregunte constantemente en qué estado se encuentra y en base a éste realizar una transición.

2.1 Clases Involucradas

Configuración: con el fin de que los intervalos de tiempo y transiciones sean configurables, esta clase es encargada de levantar la información que contiene el archivo de configuración y establecerla como parte de sus atributos.
Fantasma: encapsula el comportamiento que va a tener el fantasma en el juego, en el cual puede morir, revivir y convertirse en presa.
Estado: interfaz en común que implementan los distintos tipos de estado.
Cazador: modela la clase en la cual se incrementa la ira según el tiempo transcurrido, puede convertirse en presa y no puede ser comido.
Pasivo: clase abstracta en la cual no puede ser convertido a presa ni puede incrementar su ira. A la vez, luego de un tiempo dependiente del estado, pasa a ser cazador.
Presa: clase que representa al estado en el cual un fantasma puede ser comido.
Muerto: clase en la cual se reinicia el nivel de ira y luego de un tiempo determinado hace que el fantasma reviva como cazador.
Enemigo: clase abstracta que hereda de personaje y tiene en cuenta que todos los enemigos deben ser convertidos en presa en determinada circunstancia del juego.
Personaje: clase abstracta que define los métodos abstractos revivir() y matar() que deben implementar todas sus clases hijas.
IncrementadorDeIra: hereda de la clase TimerTask y se encarga de incrementar la ira de un estado luego del tiempo correspondiente según la configuración del juego.
Revividor: hereda de la clase TimerTask y se encarga de revivir a un personaje luego del tiempo correspondiente.

2.2 Diagrama UML



2.3 Sintaxis del archivo de configuración

Para guardar los datos de configuración se utilizó un archivo con formato JSON:

```
{
	"tiempoMuerto": 1000,
	"tiempoPresa": 1000,
	"tiemposCazador":[
	1000,
	1000,
	1000
	]
}
```

3. Extractos de código de ejemplo

3.1 Ejemplo de transición de estado en clase Fantasma

Para cada transición se crea una instancia del estado correspondiente:

```
@Override
public void matar() {
	this.estado = new Muerto(this);
}

@Override
public void convertirEnPresa() {
	this.estado = new Presa(this);
}

@Override
public void revivir() {
	this.estado = new Cazador(this);
}
```

3.2 Ejemplo de uso de la clase configuración

El constructor de la clase Cazador llama al método configurarCazador() de esta misma clase, el cual utiliza la clase Configuración para determinar los tiempos en que se produce un incremento de ira.

```
private void configurarCazador() {
	this.tiemposDeEspera = new ArrayList<>();
	this.cantidadNivelesIra = 3;
```

```
	Configuracion configuracion = Configuracion.getConfiguracion();
	for (Long tiempo : configuracion.getTiemposCazador()) {
		tiemposDeEspera.add(tiempo);
	}
}
```

Por otro lado, en la clase Muerto, se utiliza la clase Configuración para obtener el tiempo en el que el fantasma debe revivir.

```
private static int obtenerTiempoMuerto() {
	return Configuracion.getConfiguracion().getTiempoMuerto().intValue();
}
```

4. Conclusiones

En conclusión, el diseño provisto logra cumplir con las reglas propuestas en el enunciado, mientras a su vez, permite en un principio extender cada funcionalidad y agregar nuevas entidades a medida que vayan siendo necesarias para completar el sistema.

Trabajo Práctico Pacman 75.10
Primera Iteración

Implementación del Pacman 75.10 - Primera iteración

1 Implementación pedida

1.1. Modelo de dominio

En esta primera iteración se pide diseñar e implementar solamente un fantasma del juego, contemplando el tiempo que transcurre que afecta el comportamiento del fantasma.
Se pide modelar dejando de lado temas de performance por el momento.

El fantasma del juego está caracterizado por poder estar:

• Muerto
• Como fantasma cazador, con niveles de agresividad
• Como fantasma Presa

El fantasma se puede mover, puede ser comido, puede incrementar su ira. Por naturaleza al comienzo es cazador y puede ser convertido en presa en cualquier momento. Siendo cazador, cada X1, X2 y X3 intervalos incrementa su ira, luego de X3 permanece en ese nivel de ira o agresividad.

Si un fantasma esta muerto no puede incrementar su ira. No puede ser comido ni tampoco puede ser convertido en presa. Al morirse debe reiniciar el nivel de agresividad y en un tiempo X4 debe volver a "revivir" como cazador para comenzar el ciclo.

Si un fantasma es cazador no puede ser comido, puede incrementar su nivel de agresividad (hasta un máximo) y puede ser convertido en presa.

Si un fantasma es presa puede ser comido (y debe estar muerto luego de ser comido), no puede incrementar su nivel de agresividad y no puede ser convertido a presa. Al cabo de un tiempo x (ver aclaraciones) debe volver a un naturaleza cazador.

No se pide nada de implementación de interfaz gráfica ni ninguna otra funcionalidad del juego que la anteriormente descrita.

1.2. Test unitarios

Se piden un set de test unitarios sobre el modelo de dominio descrito en el punto 1.1

El set de test unitarios que se desarrolle debe representar un código valioso, debe motivar al equipo de trabajo a mantenerlo en el tiempo por la utilidad que brinda. Debe seguir las buenas prácticas de implementación de test unitarios.

Se debe contemplar un conjunto amplio y abarcativo de casos de prueba sobre el modelo de dominio.

2. Aclaraciones sobre la implementación

Si bien las siguientes etapas contemplaran como darle inteligencia de robot al fantasma, meterlo en el laberinto clásico del juego con sus bolitas (bolones y frutas) y agregar el Pacman, para esta entrega solo se pide implementar del dominio del Pacman descrito en el punto 1.1, permitiendo con con los tests validar las reglas del mismo.

- Todos los ticks o intervalos de tiempo, transiciones, etc, deben ser configurables.

3. Criterios de corrección

- El diseño del dominio
- El diseño de código
- Los test unitarios
- Cumplir con toda la funcionalidad descrita
- El informe completo. Carátula, índice, enunciado, decisiones para la resolución acompañada si es necesario de diagramas UML con descripciones orientadas a los contenidos de la materia, extractos de códigos de ejemplo, y conclusiones.

4. Herramientas a utilizar

- Maven >= 2
- JUnit 4.xx
- Repositorio Git

5. Sobre la entrega

Se debe entregar en un único archivo comprimido, el informe completo junto al repositorio local git en el que trabajaron (incluyendo el directorio .git), a través del campus.