FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

75.10 – TÉCNICAS DE DISEÑO

TRABAJO PRÁCTICO PACMAN

Tercera Iteración

2do cuatrimestre 2014

Grupo 1

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

Indice

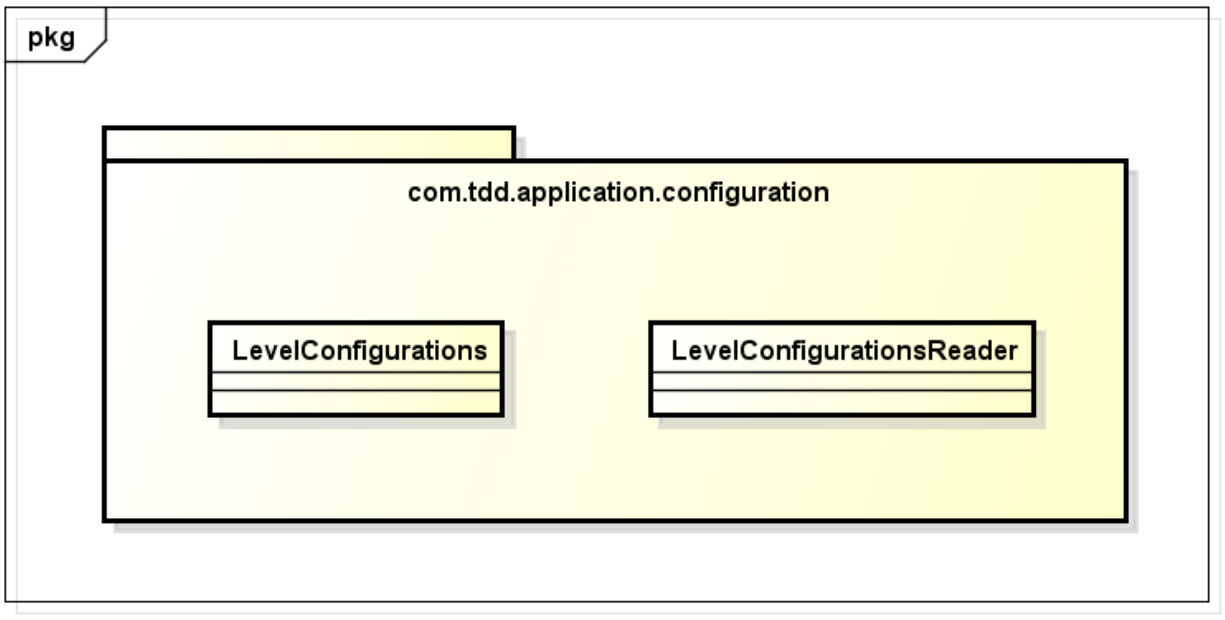
1. Objetivo

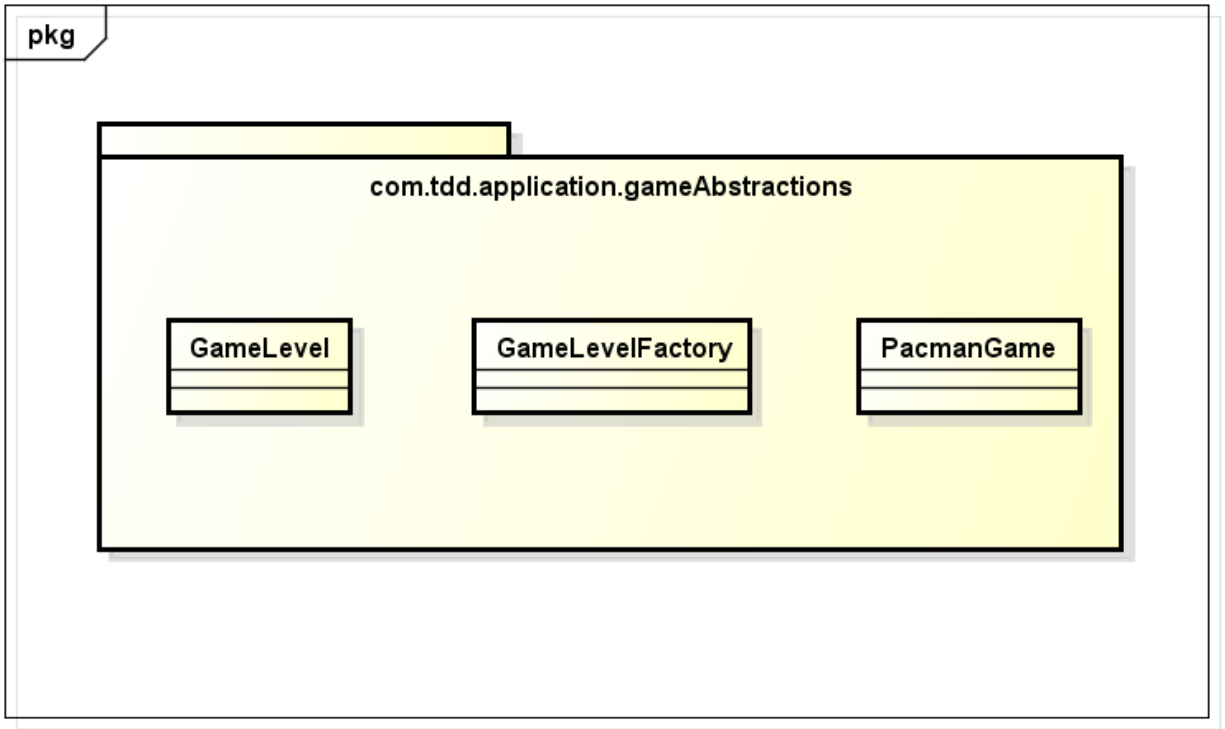
El objetivo de esta iteración es extender el modelo de dominio de la primer y segunda iteración agregando al juego una interfaz gráfica, el manejo de vidas, puntos y niveles, y las frutas como nuevo componente.

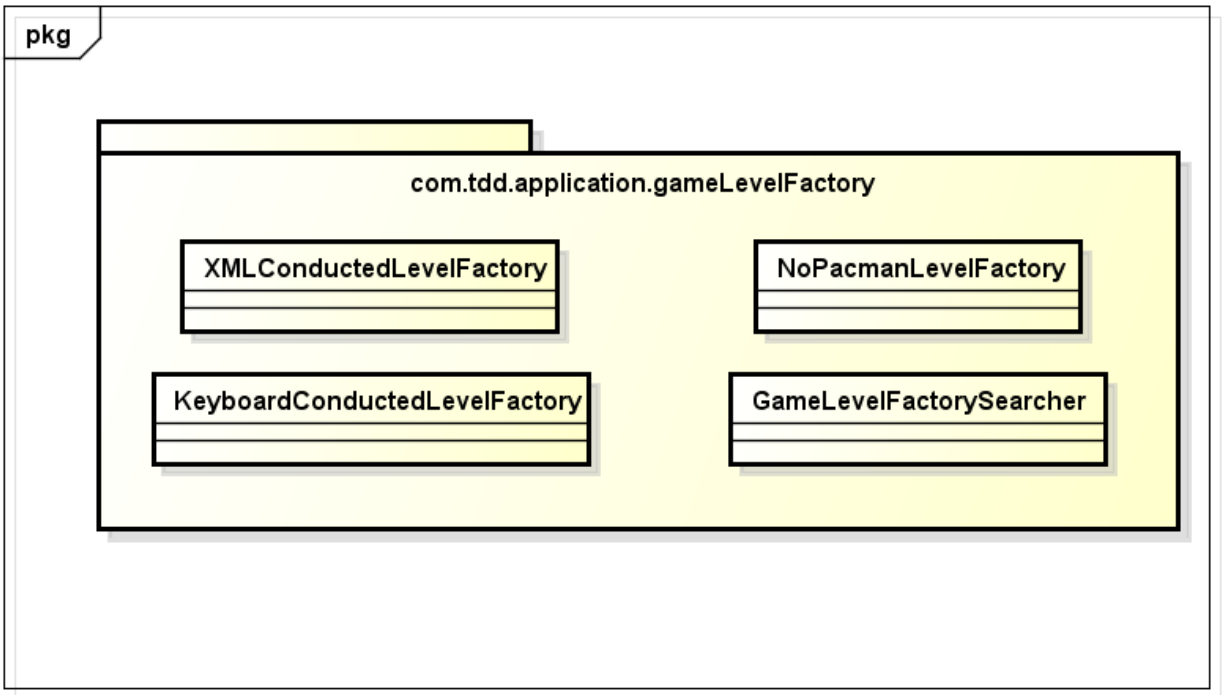
2. Decisiones para la resolución del problema

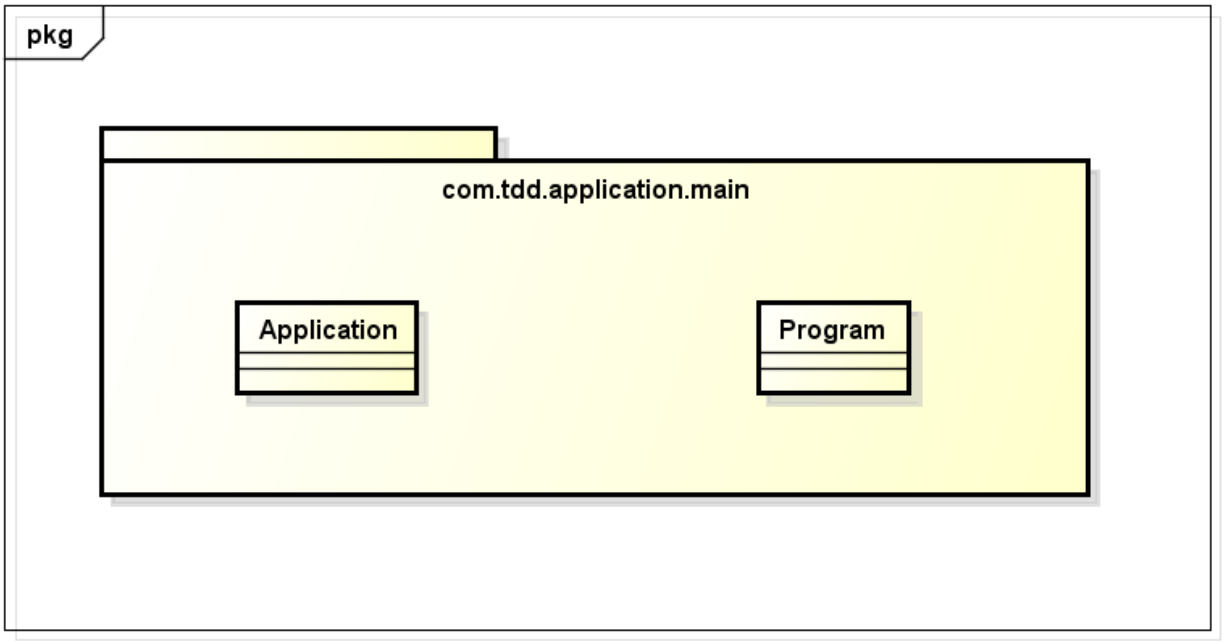
La interfaz gráfica que se decidió usar es Swing, ya que por un lado es nativa de java y eso evita complicaciones en la instalación de librerías y por otro lado, se probó previamente el manejo de la librería slick2d con resultados insatisfactorios por contar esta con su propio bucle donde corre el juego. Con lo cual se complicaba la separación de las vistas del modelo.
A la vez, se decidió usar un Thread exclusivo para el funcionamiento del gameloop ya que su iteración constante dificultaba el funcionamiento de la interfaz gráfica y la lectura de teclado. Como consecuencia, hubo que manejar temas de concurrencia en cuanto al uso de recursos como listas que usaban los dos hilos a la vez.
Para la incorporación de las frutas al juego se utilizó una nueva estrategia que usa el movimiento aleatorio definido en la iteración anterior. Las vidas y puntos se incorporaron dentro de la clase Pacman.

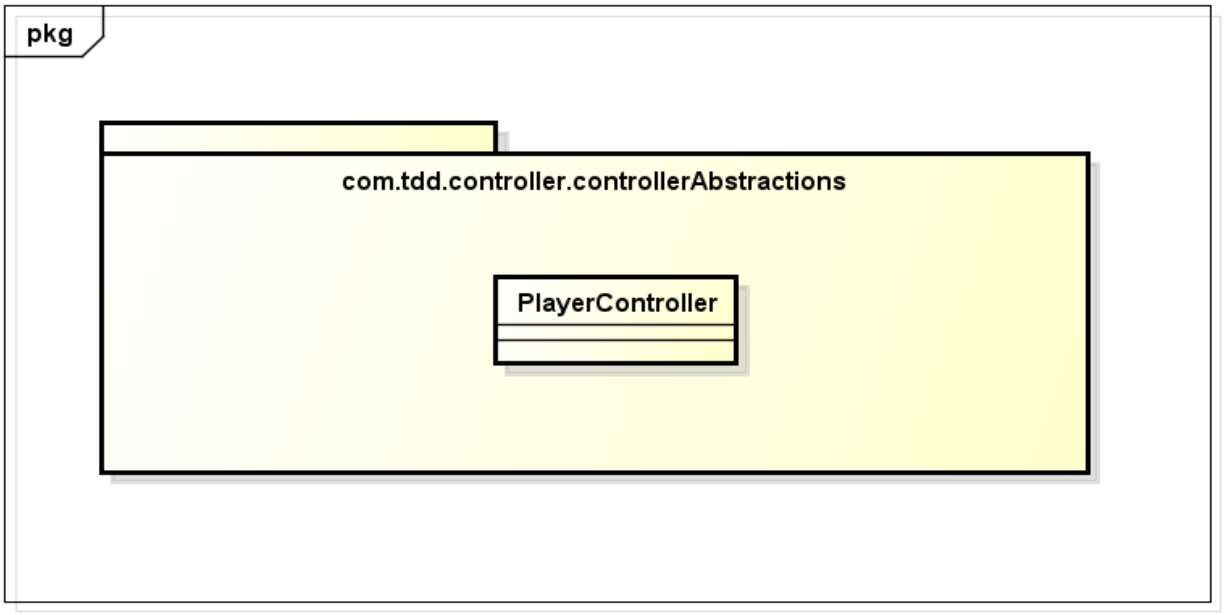
2.1 Clases y Paquetes Involucrados

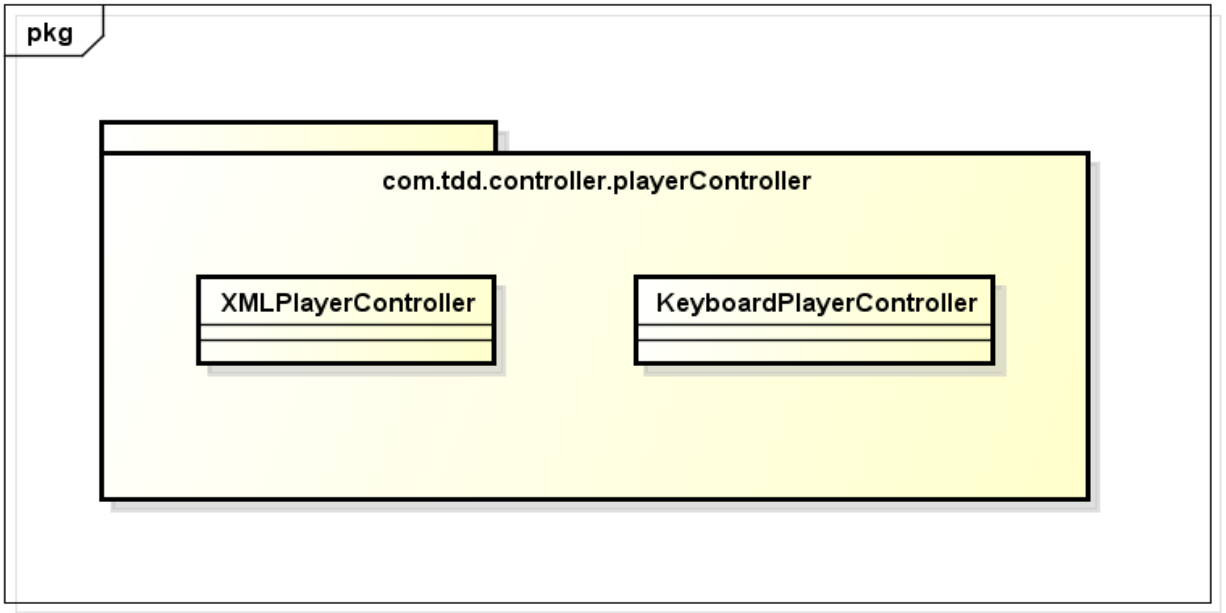


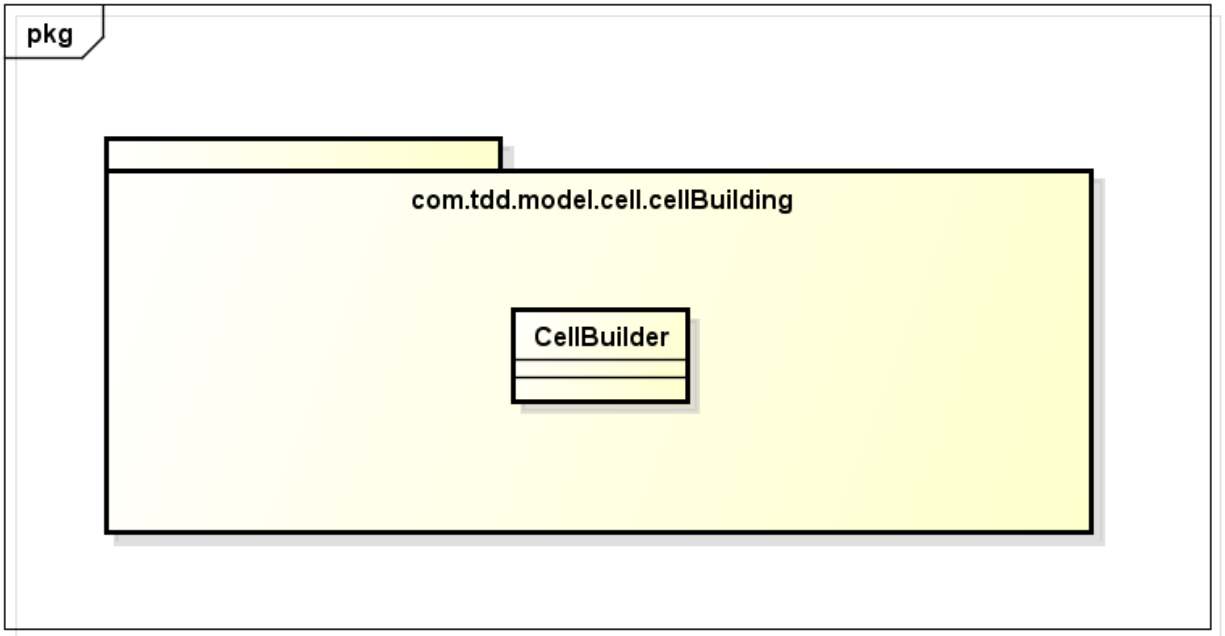


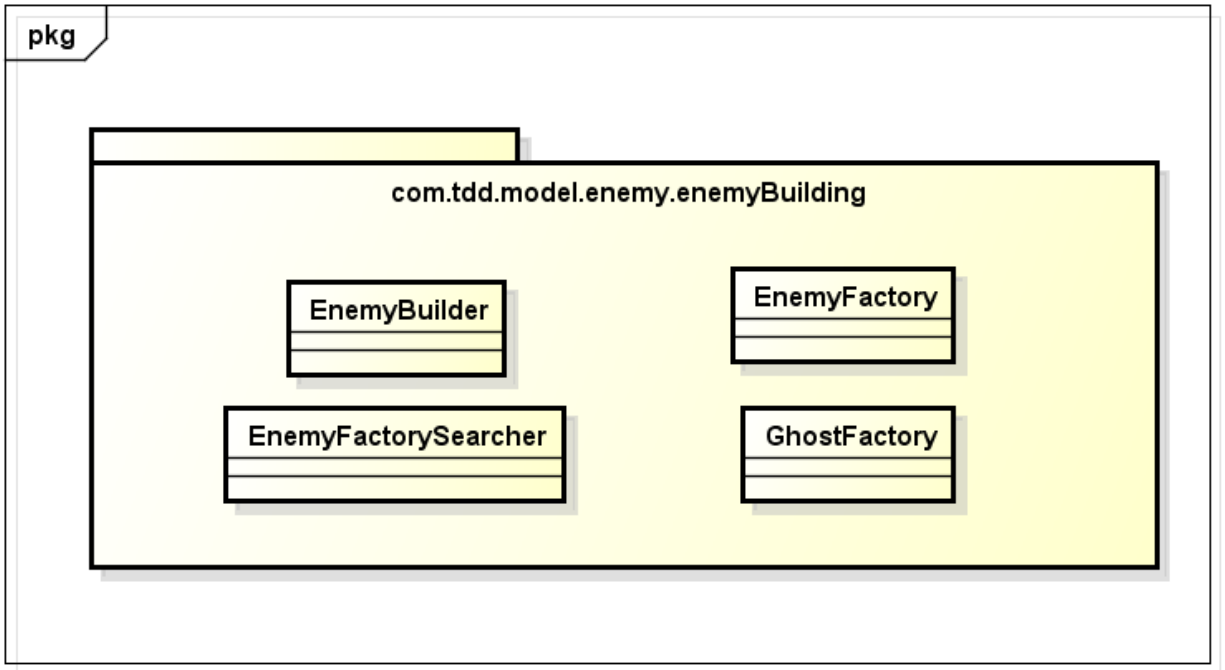


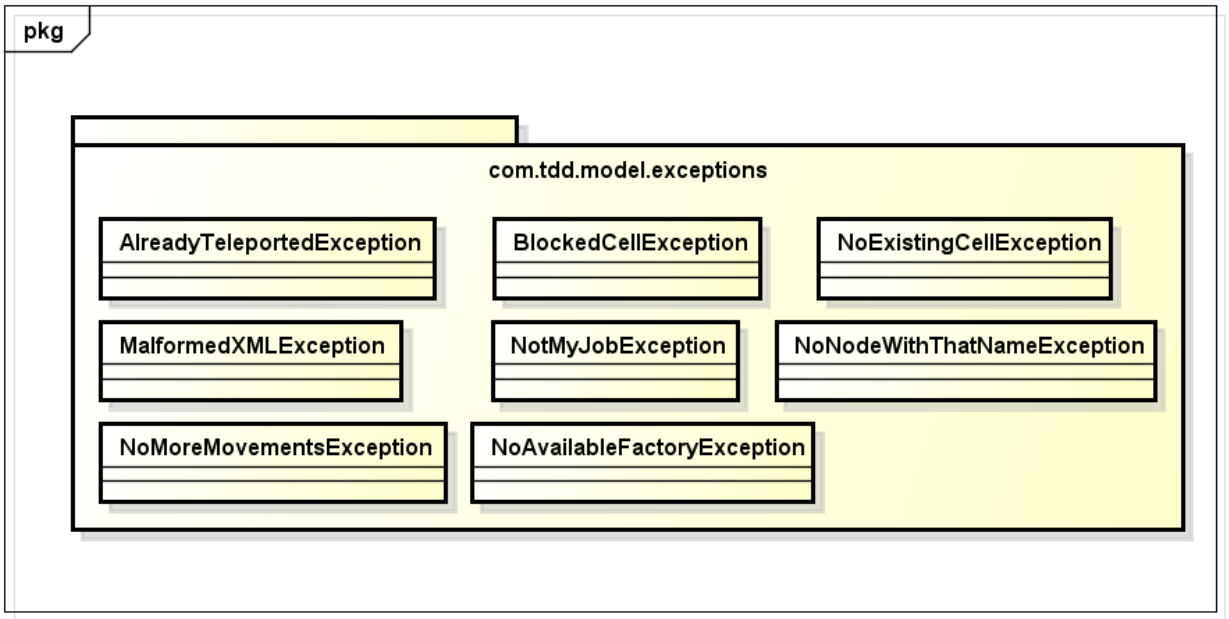


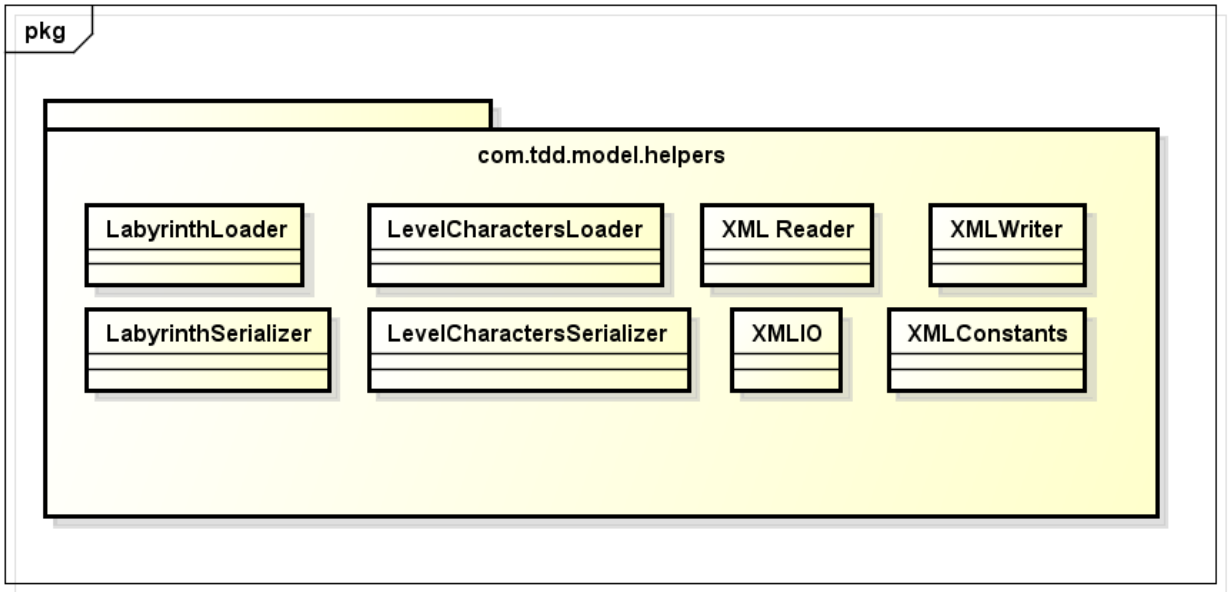


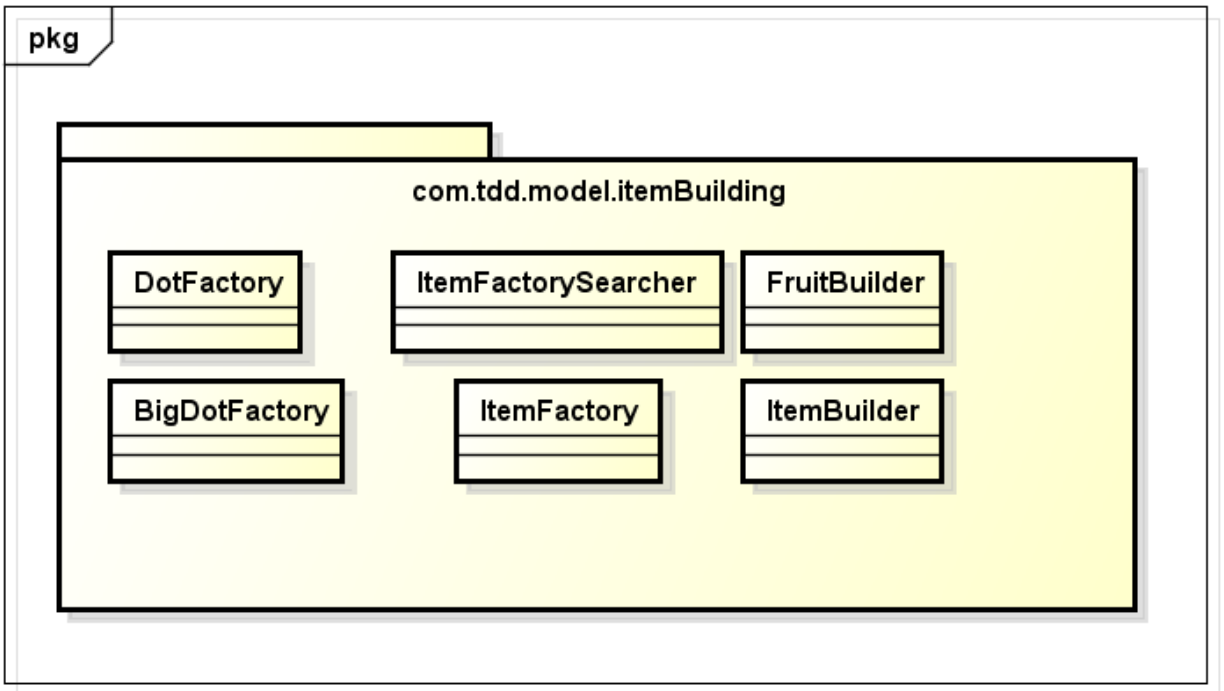


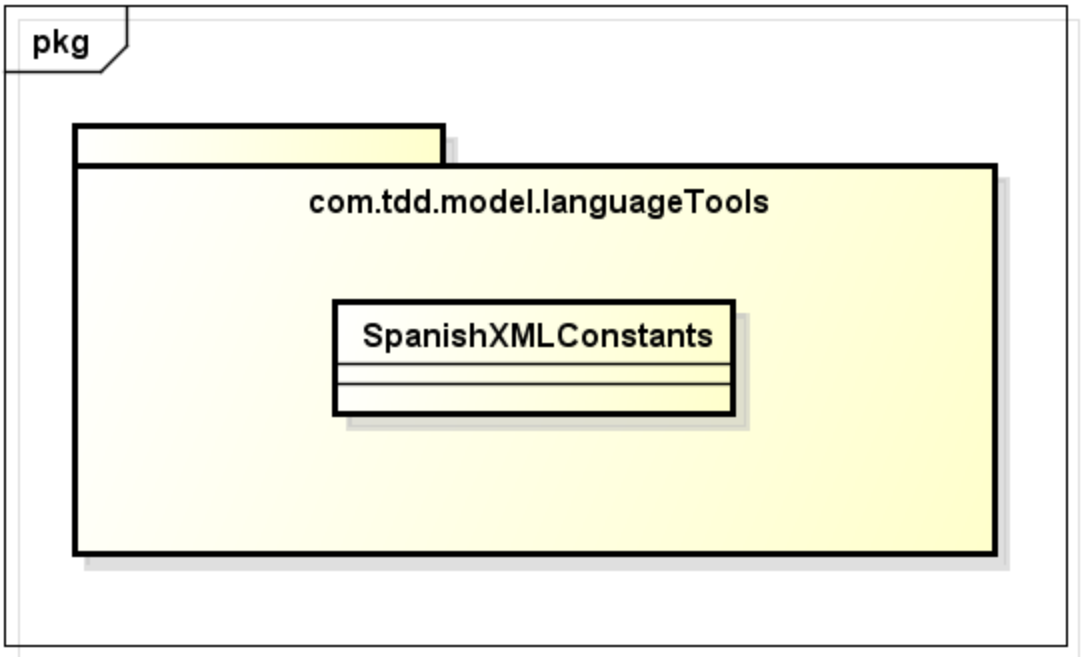


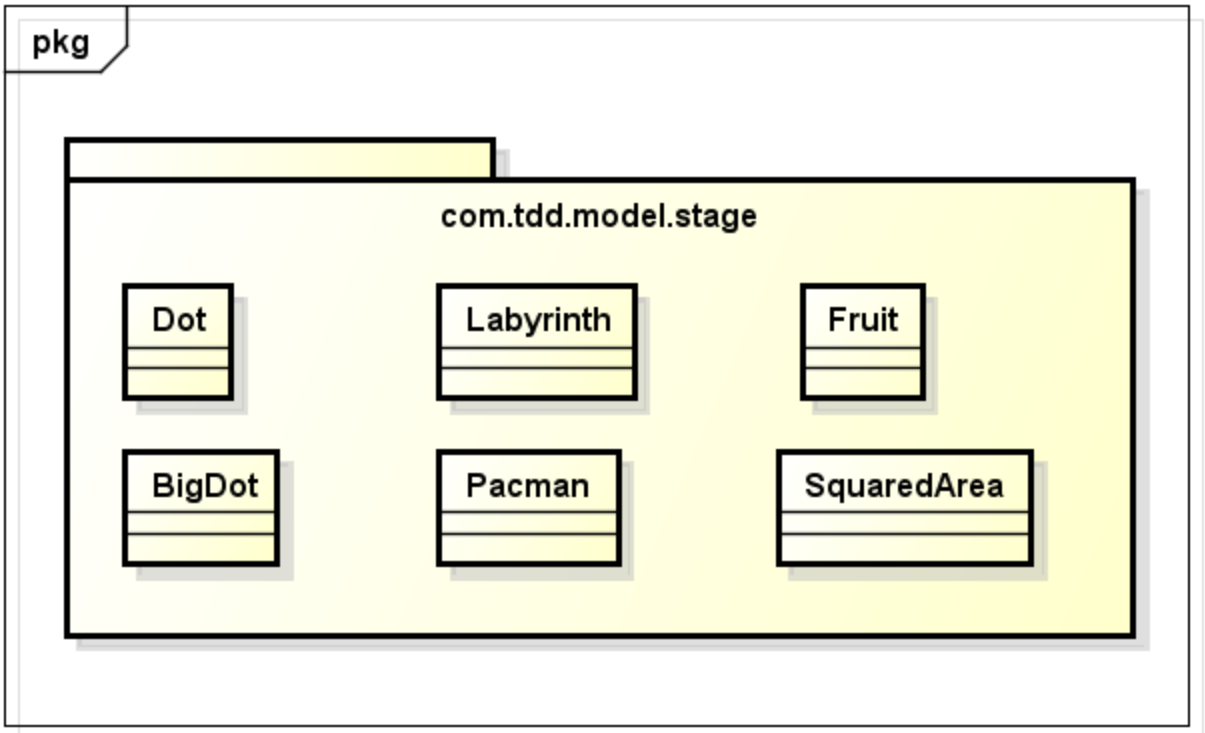


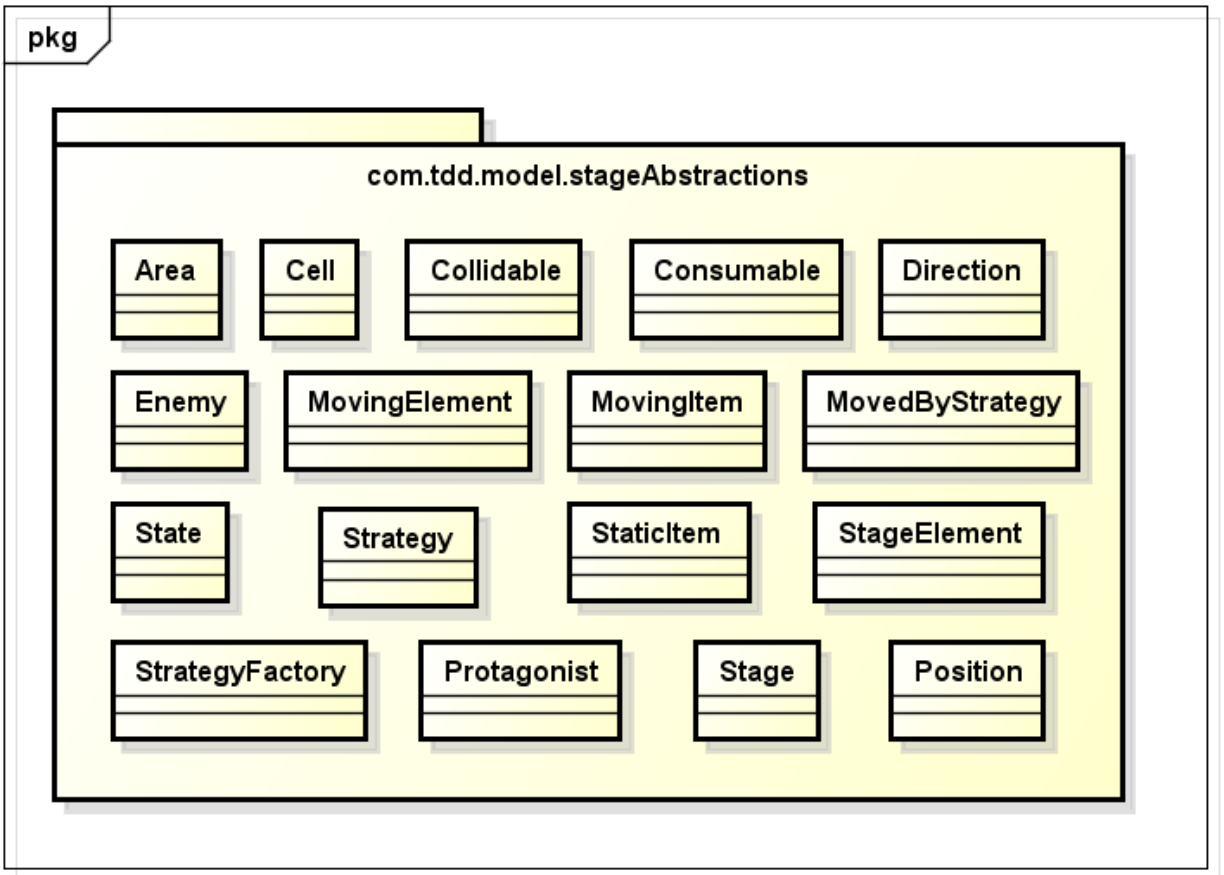


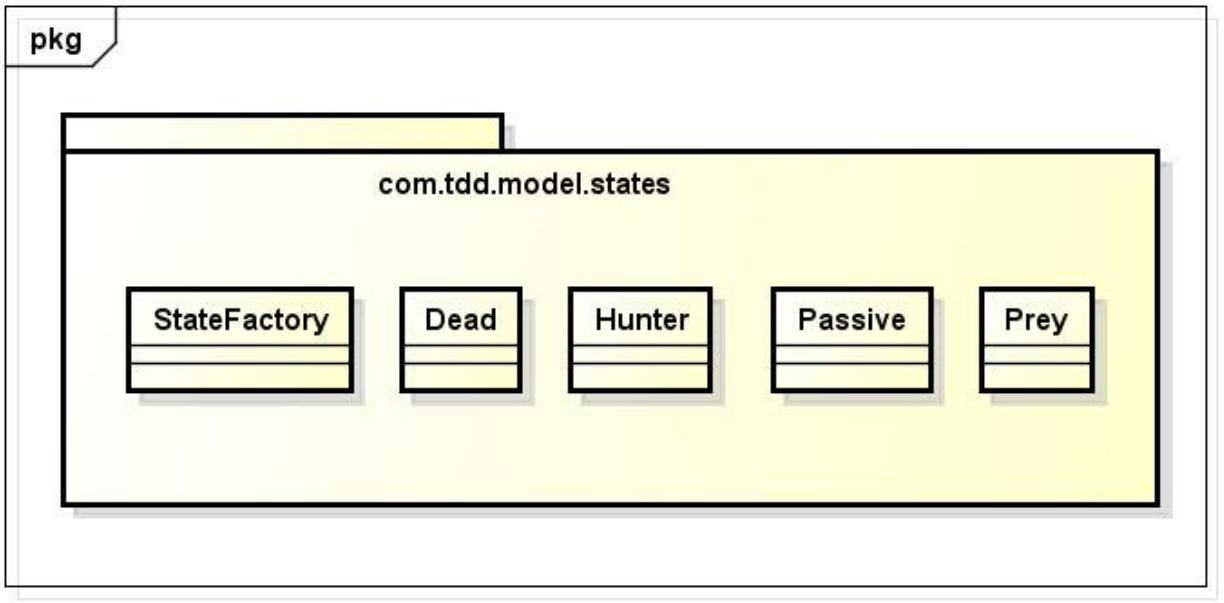


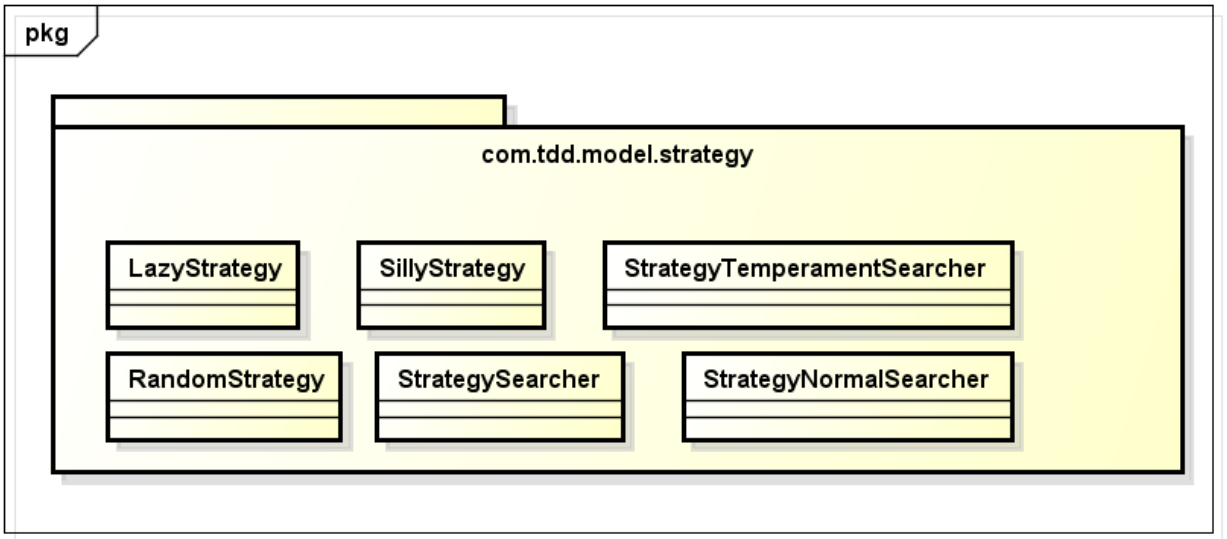


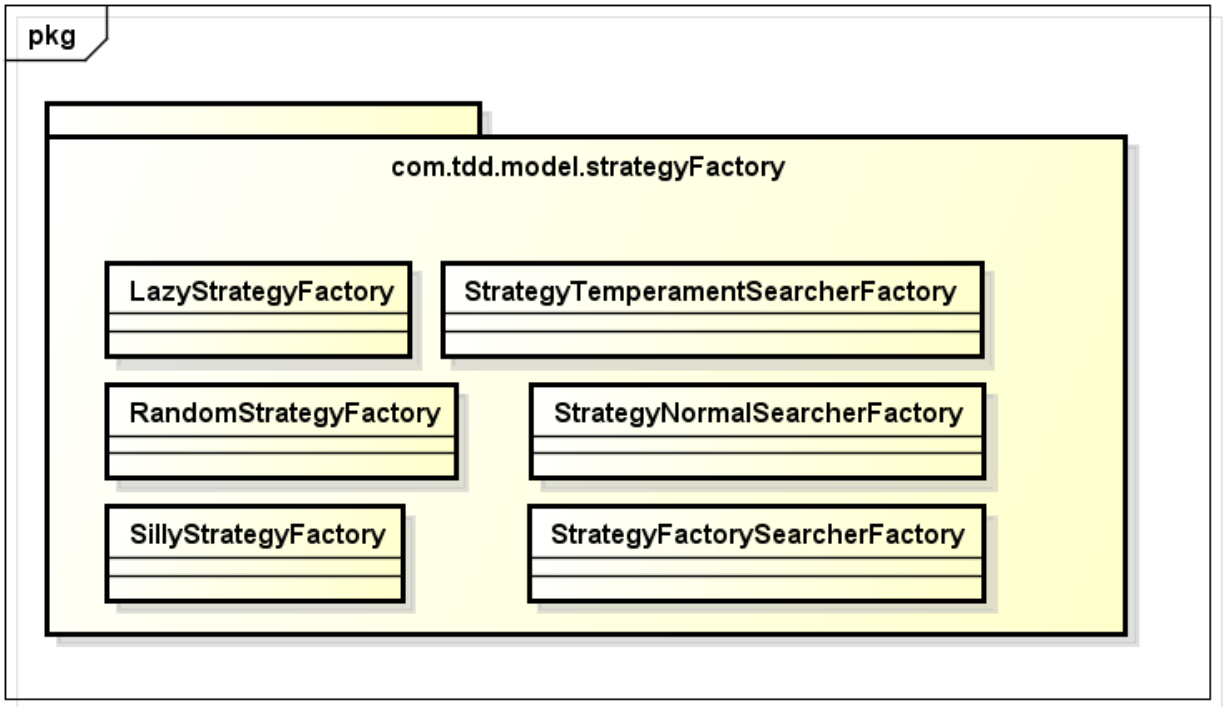


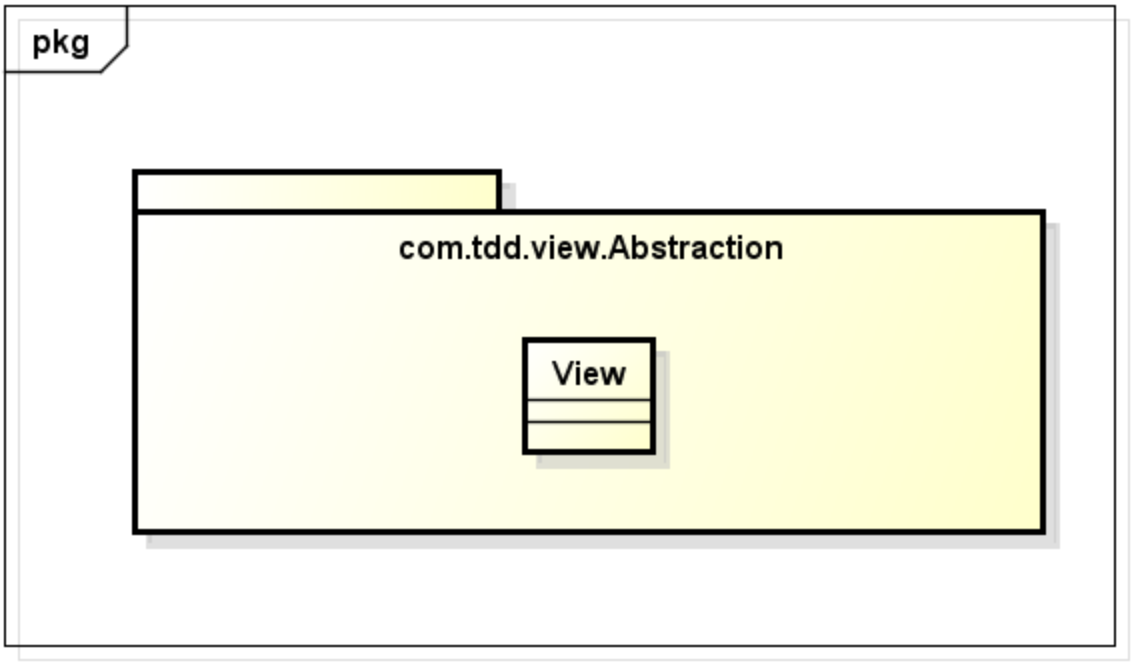


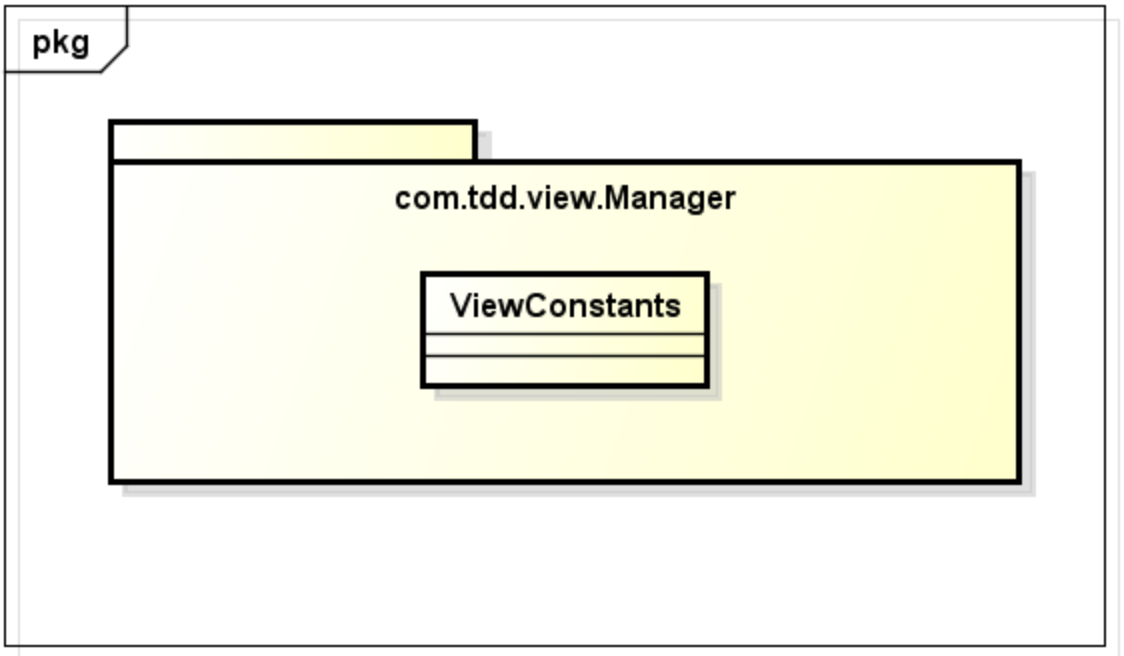


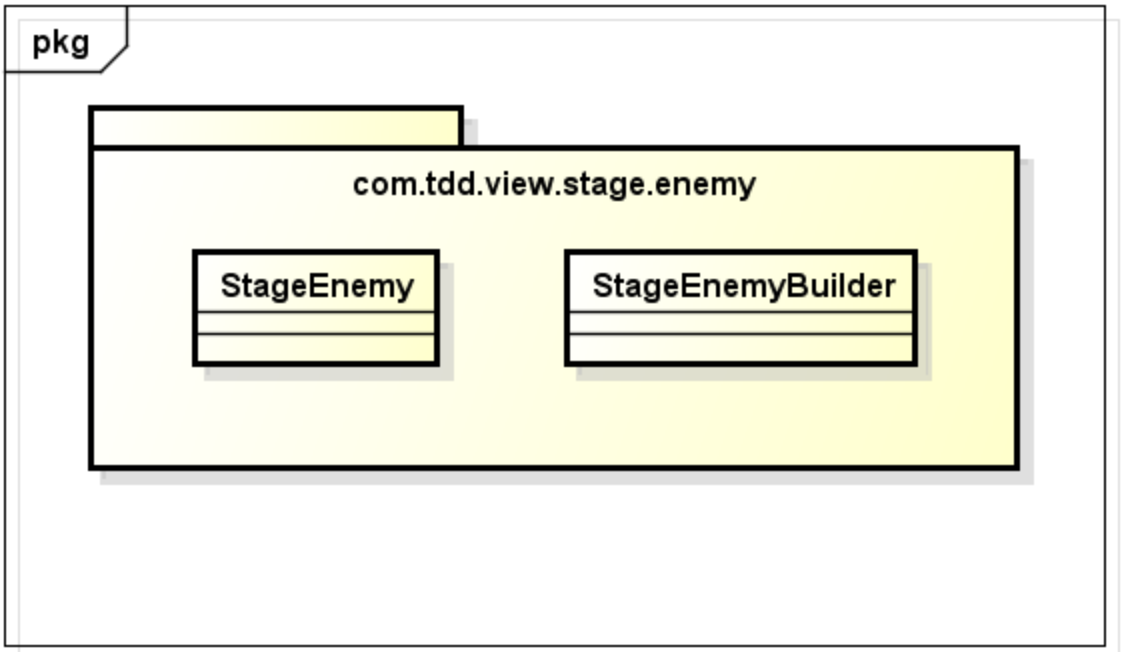


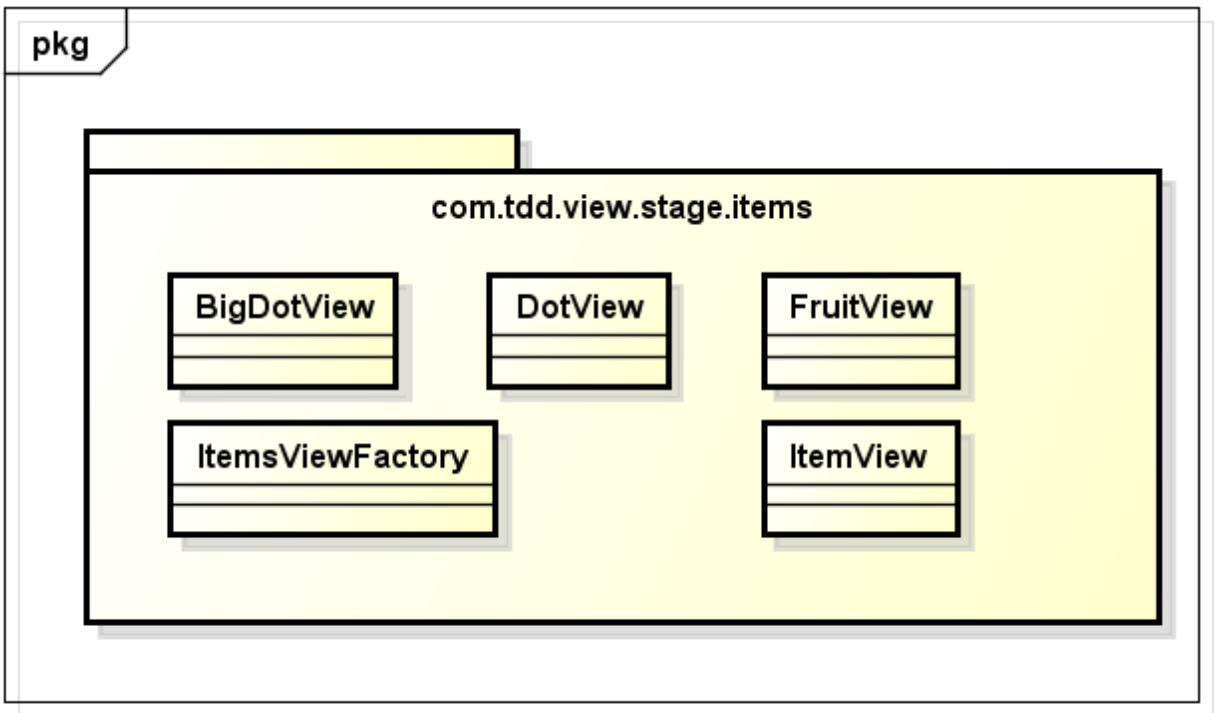


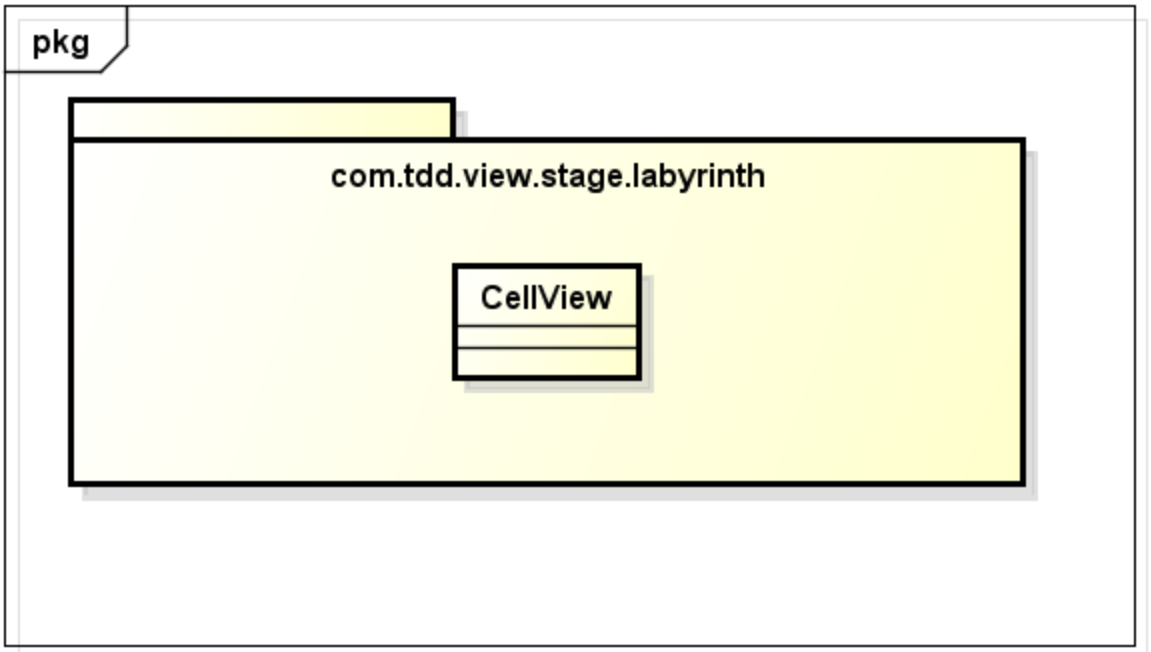


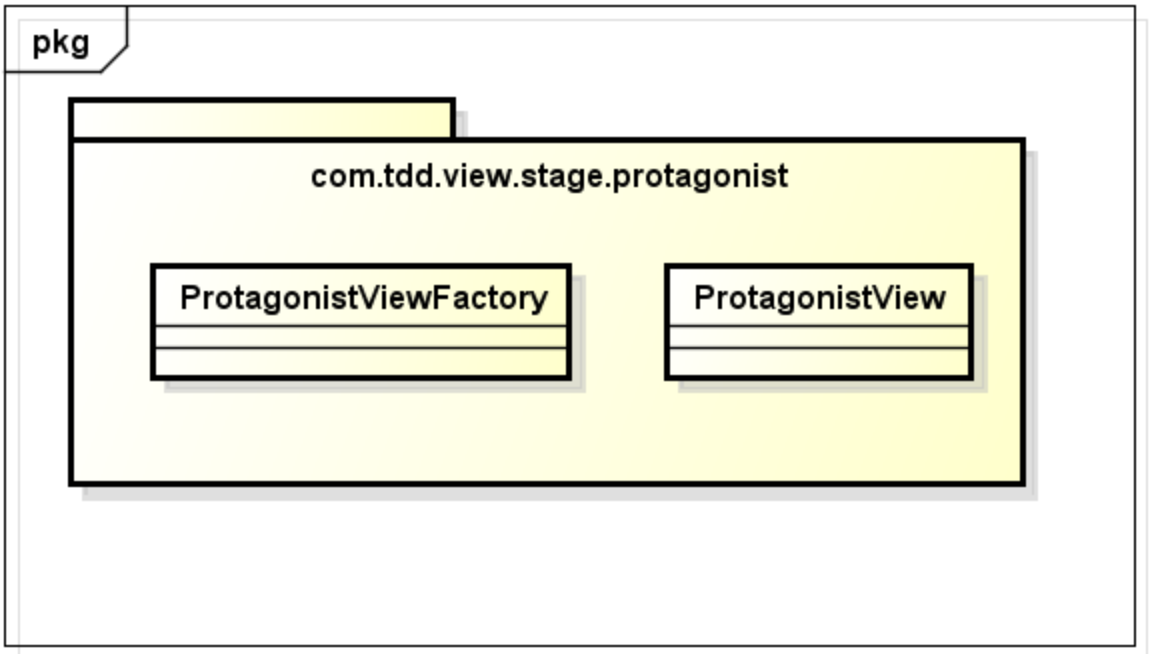


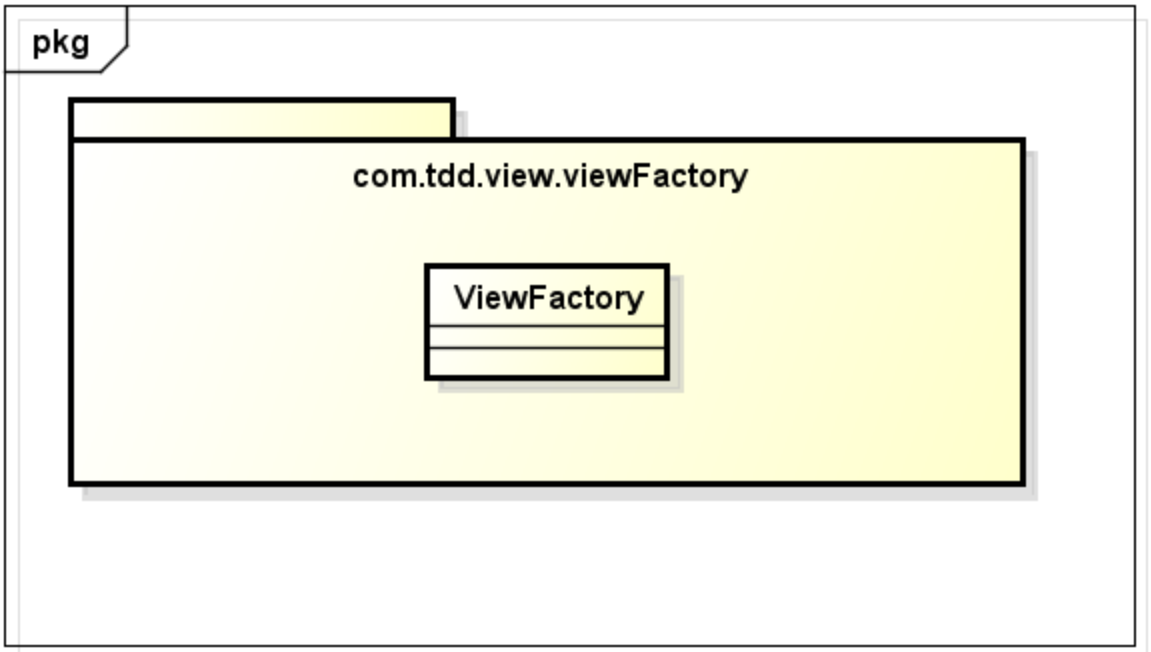


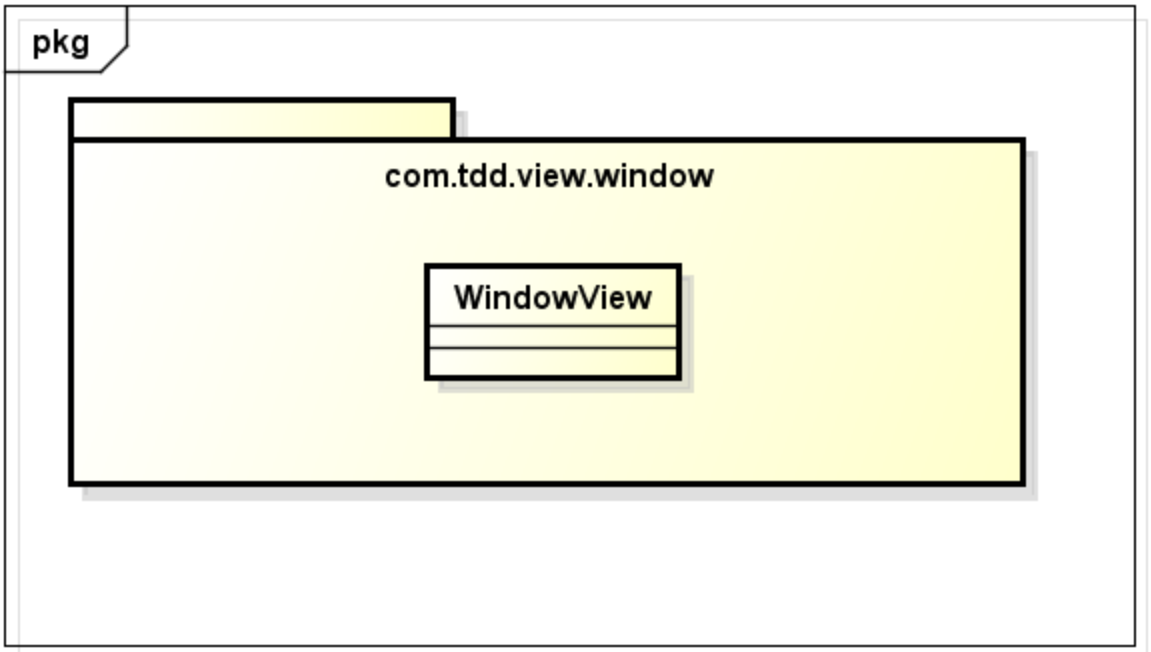












3. Ejecución del programa

Para iniciar el programa es necesario que reciba como argumento la ruta del archivo xml con el cual se obtienen los datos de configuración del juego.

Por ejemplo : ***./juegos/juego2/juego3.xml***

4. Extractos de código de ejemplo

4.1 Gameloop en clase GameLevel

```
public void gameLoop() {
        while (this.protagonist.isAlive()) {
                this.viewManager.addController(keyboardController);
                int i = 0;
                while (i < this.levelFactories.size() && this.protagonist.isAlive()) {
                    try {
                        GameLevelFactory levelFactory = this.levelFactories.get(i);
                        GameLevel level = levelFactory.createLevel();
                        level.setViewManager(this.viewManager);
                        level.populateWithProtagonist(this.protagonist, this.keyboardController);
                        level.levelLoop();
                    } catch (MalformedXMLException ex) {
                        Logger.getLogger(ActualGame.class.getName()).log(Level.SEVERE, null, ex);
                    }
                ++i;
                this.viewManager.reset();
                }
        }
}
```

3.2 Thread utilizado para llamar al loop en clase Application

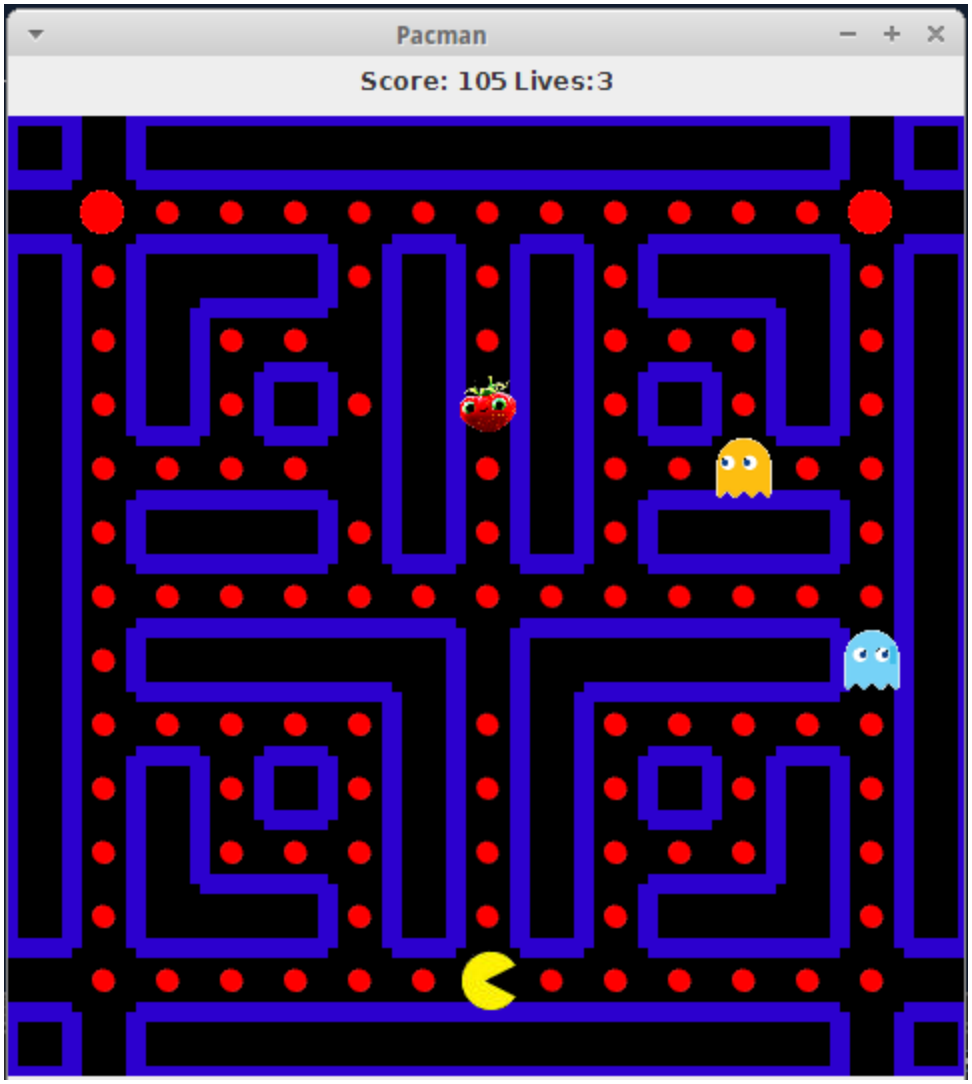
```
public void run() {
        this.view.showWindow();
        MyThread thread = new MyThread(this.game);
        thread.start();
}

private class MyThread extends Thread {
        private PacmanGame myGame;
```

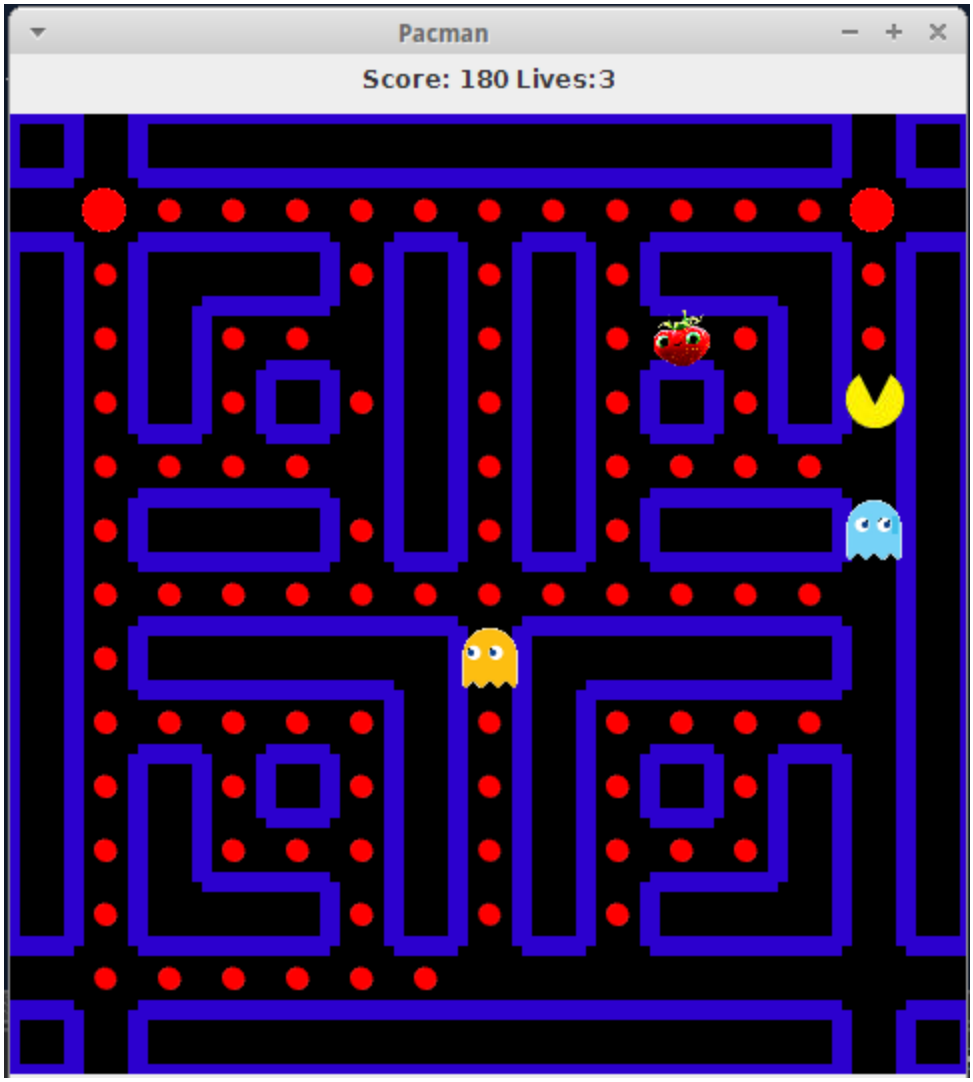
```
	public MyThread(PacmanGame givenGame) {
		this.myGame = givenGame;
	}
	@Override
	public void run() {
		this.myGame.gameLoop();
	}
}
```

5. Capturas de Pantalla

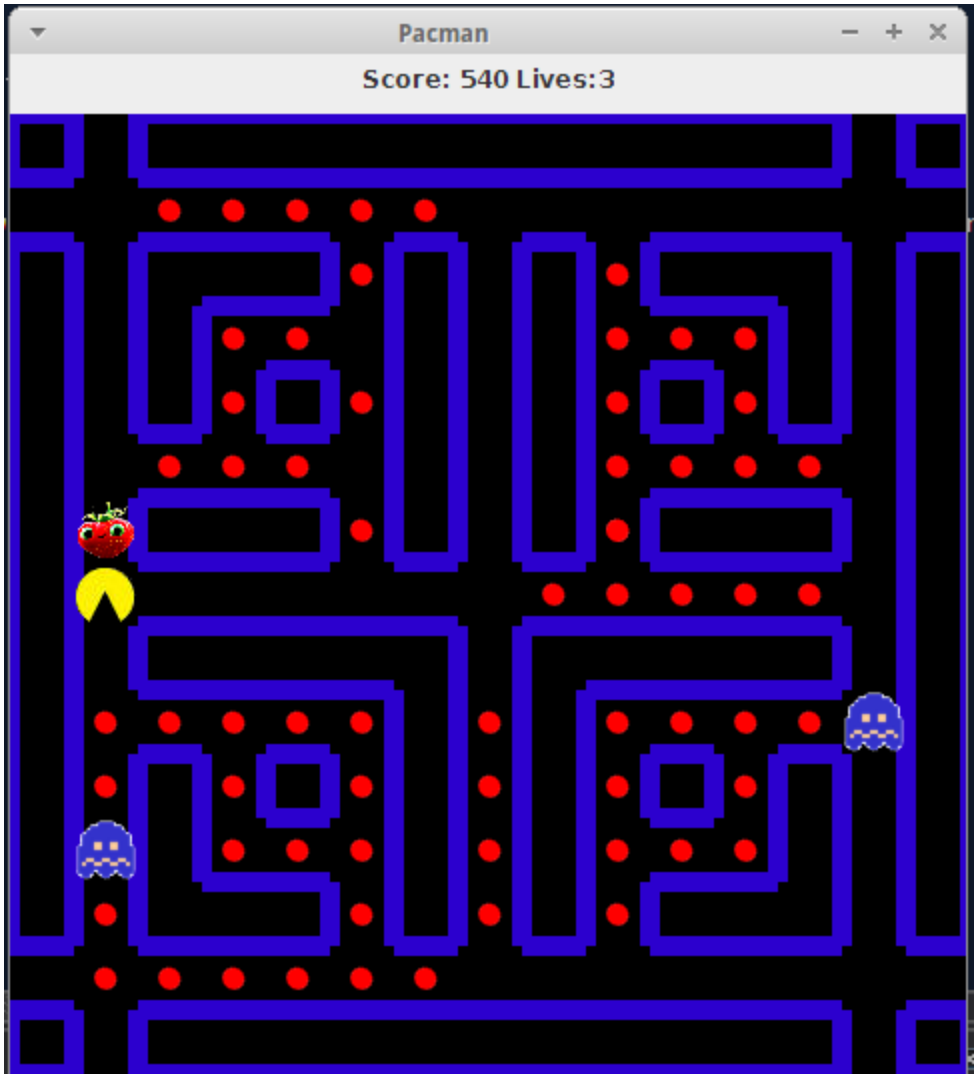
5.1 Inicio del Juego



5.2 Fantasmas como cazadores persiguiendo al Pacman

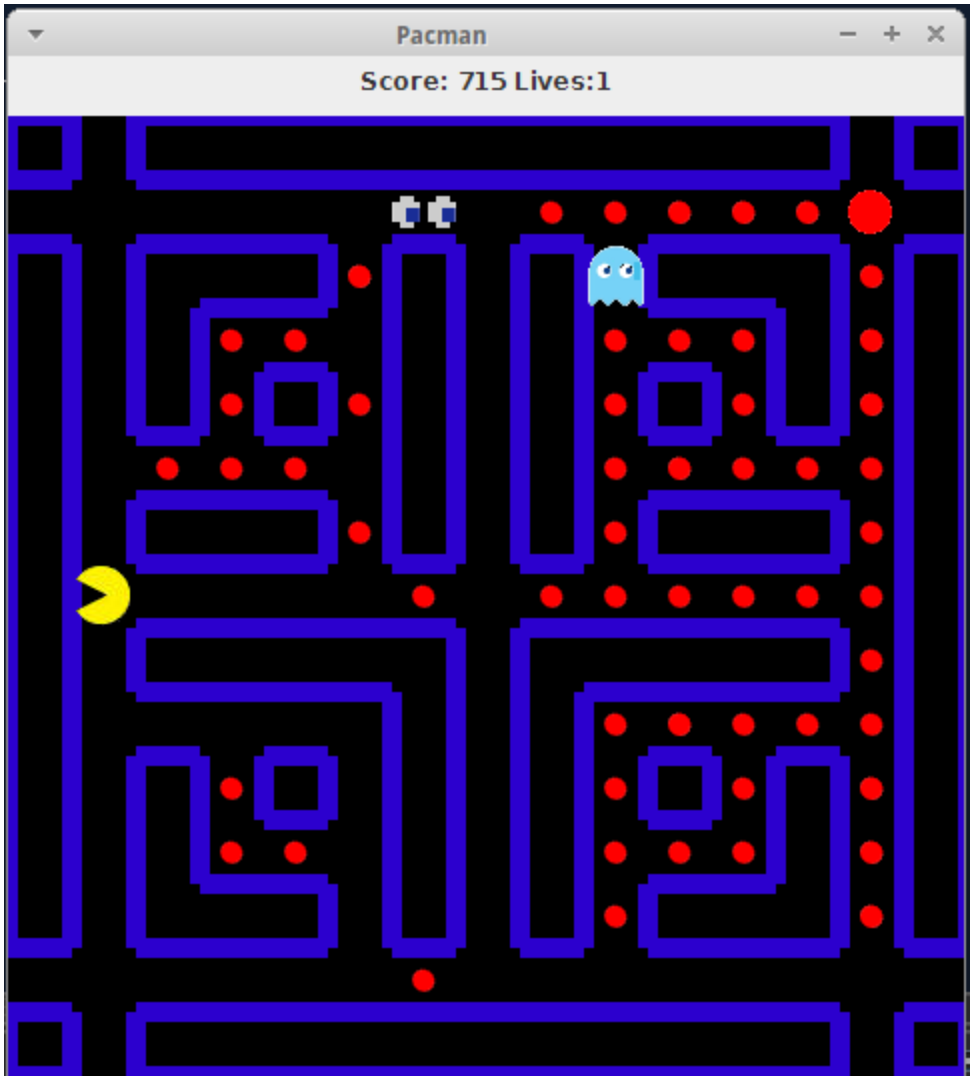


5.3 Fantasmas como presa siendo perseguidos por el pacman



5.4 Fantasma en modo cazador y fantasma en modo muerto

Cuando está muerto, solo se ven los ojos y su estrategia de movimiento cambia a aleatoria para que no siga persiguiendo al Pacman y lo coma cuando revive.



6. Conclusiones

Si bien hubo muchos cambios y refactorizaciones a lo largo de todo el desarrollo, fue de vital importancia haber implementado ciertos patrones, tanto de arquitectura como de diseño, para que, junto con las pruebas unitarias, las modificaciones y extensiones de código y clases puedan hacerse de forma sencilla y rápida, afectando sólo sectores puntuales de la aplicación.
Patrones de arquitectura utilizados: MVC y Layers.
Patrones de diseño utilizados: factory, state, strategy, factory method, template method, delegation (uso de helpers), mediator (los loaders y serializers), observer y double dispatch.
A la vez, la división de paquetes diseñada pretende separar a las clases abstractas que se relacionan entre sí en un paquete de clases abstractas y por otro lado definir en distintos paquetes las clases hijas de cada una de estas. De esta forma, se logra definir desde lo más general hasta lo más específico.