75.10 Técnicas de Diseño

Trabajo Práctico

1. Diseño

1.1 Introducción

El corriente trabajo consistió en diseñar e implementar todo lo necesario para tener como resultado una réplica del antiguo juego Pacman.

Para hacer esto posible se modelaron ciertas clases que representan los moldes para la creación de elementos del juego, y abstracciones que permiten tener un diseño c

.

1.2 Solución desarrollada

En esta sección se hará un resumen de la solución que se ha diseñado e implementado para el problema planteado.

Se tiene una clase Character de la cual heredan la clase Ghost y la clase Pacman.

El Pacman es controlado por un Controller, el cual tiene dentro suyo un InputReader. Este último se encarga de obtener del usuario la entrada necesaria para enviarle por medio del Controller un mensaje al Pacman para que sepa qué debe hacer.

Un Ghost puede tener varios estados y varias personalidades, para esto se tiene una jerarquía de clases de GhostState y Personality. Las clases derivadas de estas dos serán componentes de un fantasma (Ghost).

Cada personalidad puede variar la manera de mover al fantasma (Ghost), por lo que se modelo un conjunto de clases que derivan de MovementStrategy. Una instancia de Personality tiene una instancia de MovementStrategy como componente.

Las clases MazeBuilder y CharacterBuilder son las encargadas de construir a los personajes y al mapa del nivel desde archivos XML. Los métodos de construcción de ambas clases son llamados en la construcción de una instancia de la clase Board.

El Board es el encargado de manejar lo que ocurre durante cada iteración del juego. El chequeo de ciertos eventos tales como las colisiones y las notificaciones y actualizaciones de los objetos afectados. El Board tiene entre sus atributos:

- El pacman (Pacman)
- La fruta (Fruit)
- Los fantasmas (lista de Ghost)
- El laberinto (Maze)
-

El laberinto o Maze es una grilla de celdas. Cada celda puede tener o no una bola dentro. Los tipos de bola pueden ser grande o chica.

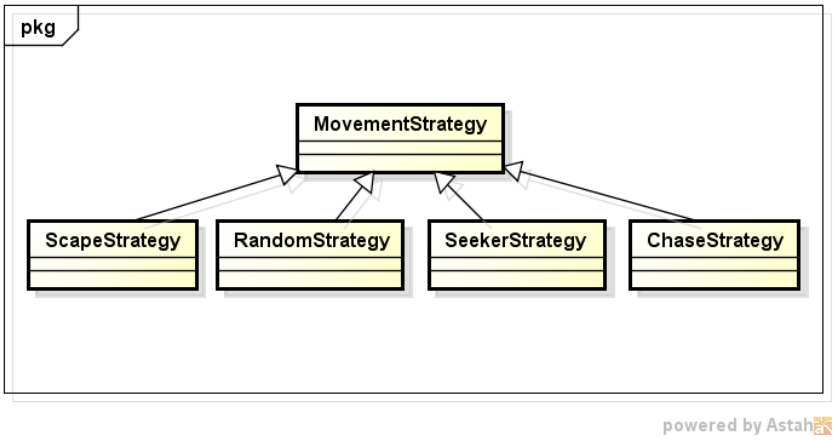
Existe un esquema basado en eventos en el cual un suscriptor se (Subscriber) se suscribe a un evento en un publicador (Publisher), y de esta manera cuando ocurre tal evento, todos los suscriptos a él son notificados.

1.3 Patrones Utilizados

Se listan aquí los patrones de diseño utilizados a lo largo del diseño de la solución.

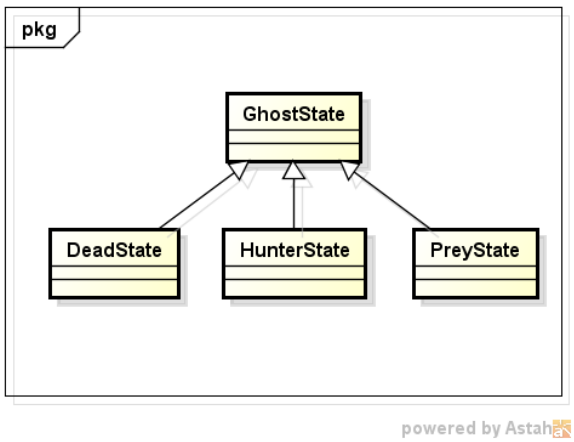
Strategy

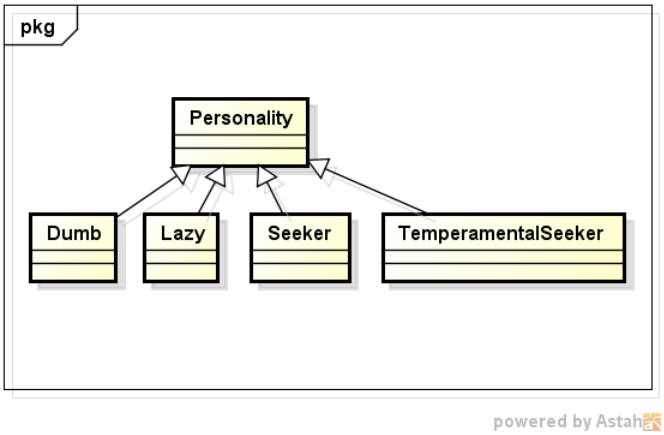
Este patrón fue utilizado para implementar las estrategias de movimiento de los fantasmas. La personalidad del fantasma posee una estrategia, y es esta la encargada de saber en dirección debe avanzar el fantasma dueño de la personalidad en el turno actual.



State

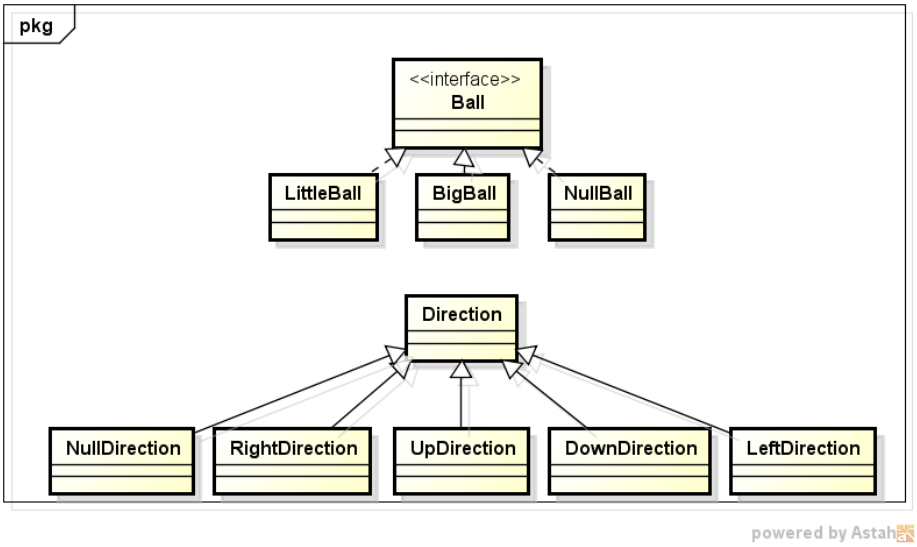
Debido a que los fantasmas pueden tener distintos estados y su comportamiento varía en base al actual, se decidió optar por utilizar este patrón. Se tiene, entonces, un fantasma que tiene una referencia a un objeto que modela un estado del fantasma. Además, este patrón es utilizado para modelar las personalidades de los fantasmas. De manera muy similar a la de los estados, un fantasma puede tener varias personalidades y su comportamiento varía según cual es la actual.





Null-Object

Este patrón, por más tonto que sea, se anota a la lista de los utilizados. Se usó en el caso, de la jerarquía de bolas (grande, chica y nula) que se diseñaron para tener distintos comportamientos para cada una, y distintas vistas asignadas a cada una.
Además, es usado en el caso de las clases derivadas de la clase dirección, teniéndose, entonces, una dirección nula. Aquel personaje que tenga como dirección actual una dirección nula no avanzará en la corriente iteración del juego.

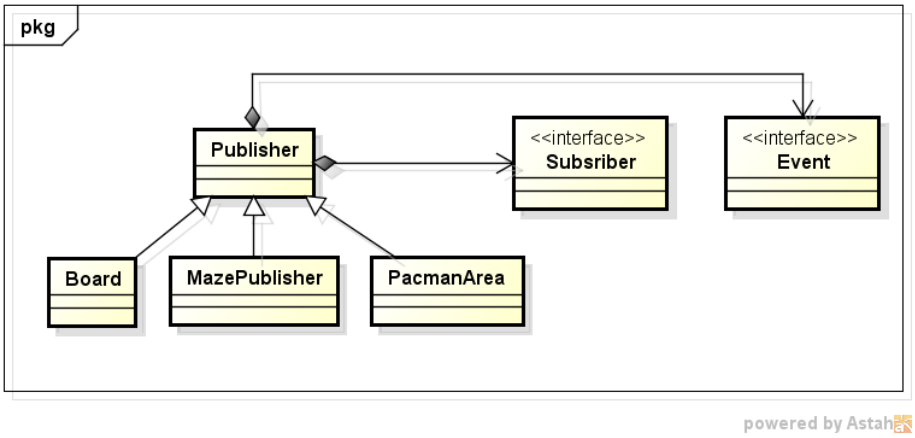


Subscriber – Publisher

En orden de tener un modelo basado en eventos se implementó este patrón, el cual consiste en una interfaz Subscriber, donde aquellos objetos que la implementan se suscriben a un evento en un objeto derivado de la clase abstracta Publisher, el cual los notificará y actualizará su estado en el momento en el que ocurra el evento al que están suscriptos. Este patrón tiene la ventaja, respecto del Observer, de que un subscriber puede estar suscripto a varios eventos distintos en un mismo publisher.

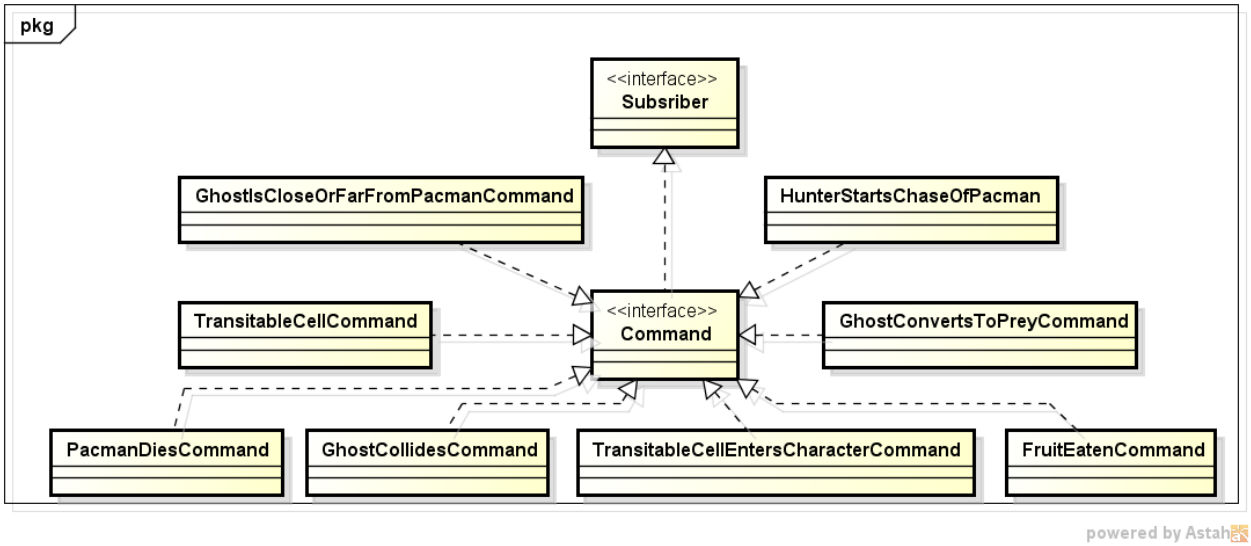
Las clases que hacen uso de este patrón son:

- Board, la cual maneja los eventos de colisión de los personajes y el de si la bola grande es comida (en tal caso los fantasmas pasan a estado presa).
- PacmanArea, la cual maneja los eventos de entrada y salida de un fantasma del área de influencia del pacman (el área dentro de la cual un fantasma puede verlo).



Command

Para poder tener encapsulado tanto el objeto que se quiere suscribir a un evento como el comportamiento que se quiere tener cuando ocurra dicho evento, se utiliza el patrón Command. En el diseño del trabajo, todas aquellas clases que son del tipo command implementan la interfaz Subscriber.



MVC

Se implementa este patrón de diseño en la solución desarrollada de manera de tener desacoplados (dentro de lo posible) el modelo, la vista y el controlador.

Factory

Se hizo uso de este patrón para la construcción de ciertos objetos del juego, tales como los que son instancias de Ghost (fantasma) y de View (Vista).

Builder

Este patrón fue utilizado para la construcción de los distintos personajes y del mapa del nivel del juego a partir de un archivo XML como entrada.

Singleton

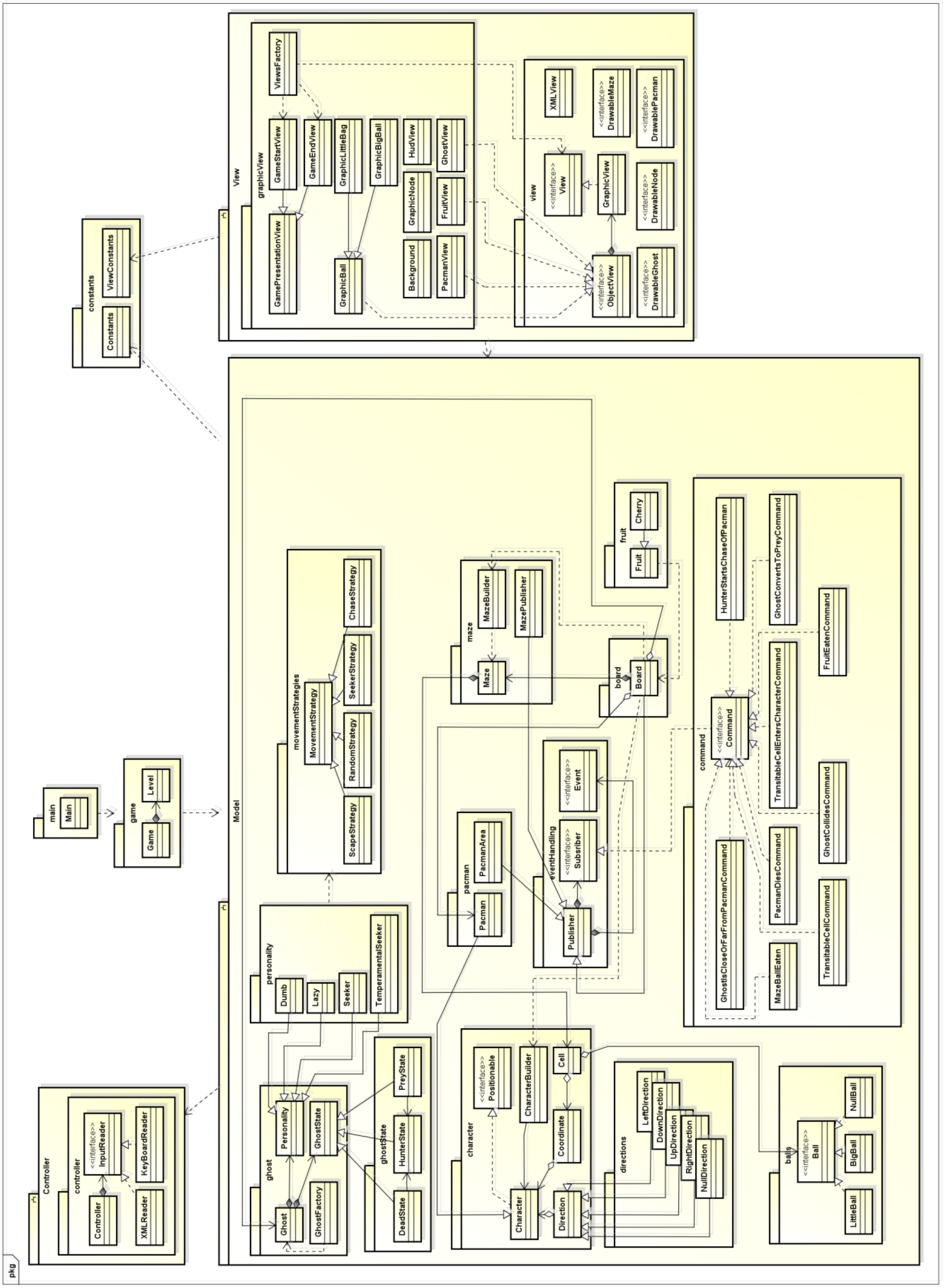
Clases singletons se tienen, la clase de constants regulares del juego, la clase de constantes de la vista, el pacman y la clase que modela el área donde puede ser visto el pacman.

Visitor (Double Dispatch)

Este patrón se utilizó en las clases derivadas de Direction para hacer la comparación entre ellas, de si son la misma o no.

1.4 Diagrama General

En la siguiente sección se observa un diagrama general de la estructura de la solución planteada. Este diagrama trata de resumir el diseño general de lo desarrollado sin entrar en detalles, dando una vista de alto nivel.


pkg
Controller
controller
<<interface>> InputReader
Controller
KeyBoardReader
XMLReader
main
Main
game
Game
Level
constants
Constants
ViewConstants
View
graphicView
ViewsFactory
GameStartView
GameEndView
GraphicLittleBag
GraphicBigBall
HudView
GhostView
GamePresentationView
GraphicBall
GraphicNode
Background
FruitView
PacmanView
view
XMLView
<<interface>> View
GraphicView
<<interface>> ObjectView
<<interface>> DrawableMaze
<<interface>> DrawablePacman
<<interface>> DrawableNode
<<interface>> DrawableGhost
Model
movementStrategies
MovementStrategy
ChaseStrategy
SeekerStrategy
RandomStrategy
ScapeStrategy
personality
Dumb
Lazy
Seeker
TemperamentalSeeker
ghost
Personality
GhostState
Ghost
GhostFactory
ghostState
PreyState
HunterState
DeadState
maze
MazeBuilder
MazePublisher
Maze
fruit
Cherry
Fruit
board
Board
pacman
PacmanArea
Pacman
eventHandling
<<interface>> Event
<<interface>> Subsriber
Publisher
command
HunterStartsChaseOfPacman
GhostConvertsToPreyCommand
FruitEatenCommand
<<interface>> Command
TransitableCellEntersCharacterCommand
GhostCollidesCommand
GhostIsCloseOrFarFromPacmanCommand
PacmanDiesCommand
TransitableCellCommand
MazeBallEaten
character
<<interface>> Positionable
CharacterBuilder
Cell
Coordinate
Character
Direction
directions
LeftDirection
DownDirection
UpDirection
RightDirection
NullDirection
balls
<<interface>> Ball
NullBall
BigBall
LittleBall
powered by Astah