FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico N° 2 — Java

AlgoThief

[7507/9502] Algoritmos y Programación III
Segundo cuatrimestre de 2021

Integrantes	Padron	Email
Francisco Duca	106308	fduca@fi.uba.ar
Selene Martinez	100439	semartinez@fi.uba.ar
Lucas Oshiro	107024	loshiro@fi.uba.ar
Federico Rubachin	96068	frubachin@fi.uba.ar
Rodrigo Vargas	97076	rvargas@fi.uba.ar

Índice

1. Supuestos **2**

2. Diagramas de clase **2**

3. Diagramas de secuencia **6**

4. Detalles de implementación **6**
4.1. Objetivo General 7
4.2. Principios de Diseño 7
4.3. Patrón de Diseño: MVC 8

1. Supuestos

En el siguiente trabajo práctico, los supuestos que tuvimos en cuenta son:

- Cuando un policia inicia por primera vez el juego, comienza siendo Rango Novato, es decir, las primeras pistas que va a recibir son pistas fáciles.
- El policia solo puede atrapar al ladron si tiene la orden de captura emitida.
- Existen edificios con matones que sirven para demorar al policia.
- A medida que el policia va resolviendo casos, en determinada cantidad su rango aumentará, por lo tanto las pistas van a ser de mayor complejidad.
- Los tipo de pistas se dividen en: pistas viajeras, pistas culturales y pistas económicas, dependiendo del edificio al que el policia visita para pedir la pista (banco, puerto/aeropuerto ó biblioteca).

2. Diagramas de clase

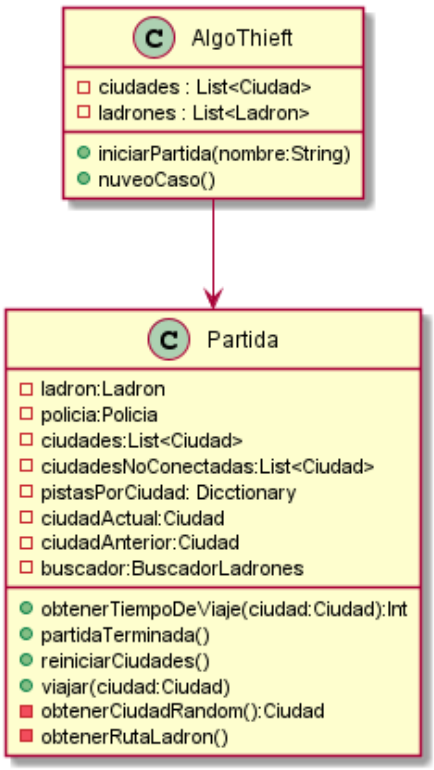

Figura 1:
En este diagrama se muestra a AlgoThieft, la clase de partida de nuestro juego en donde se iniciaran los objetos que posteriormente se comunicaran entre si.

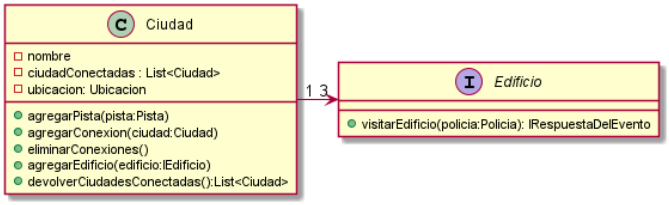

Figura 2:
Cada ciudad contara con exactamente tres edificios que se podran visitar para hallar las pistas.

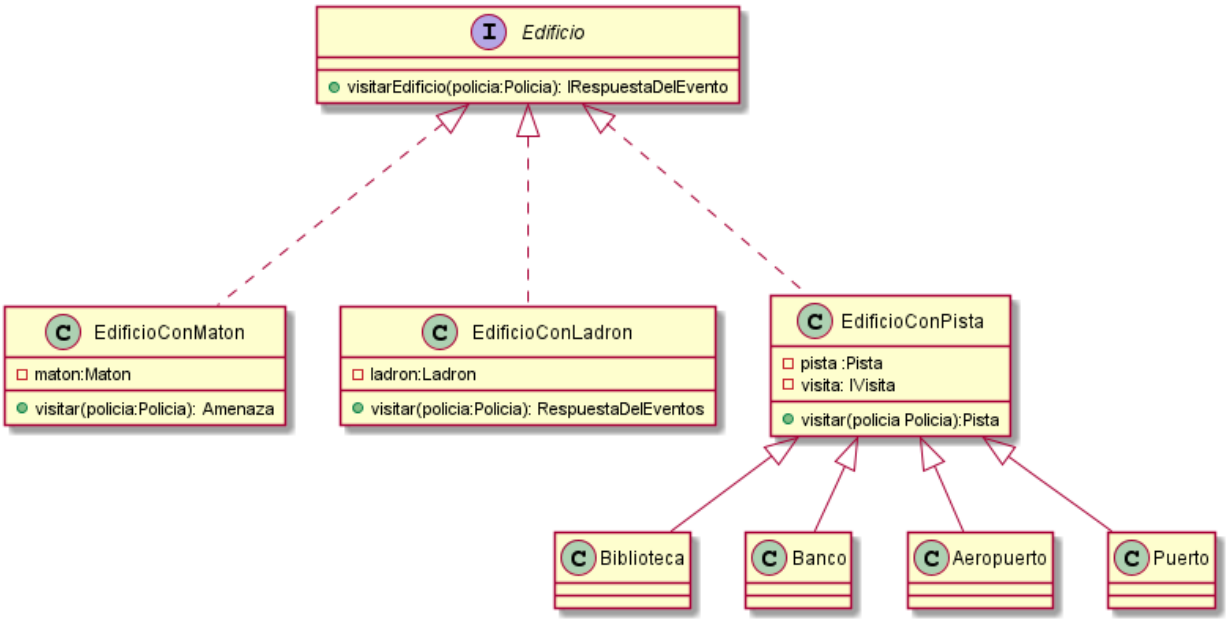

Figura 3:
A su vez cada edificio podrá ser de tres tipos distintos, dependiendo de la ciudad en que se ubique el policia.

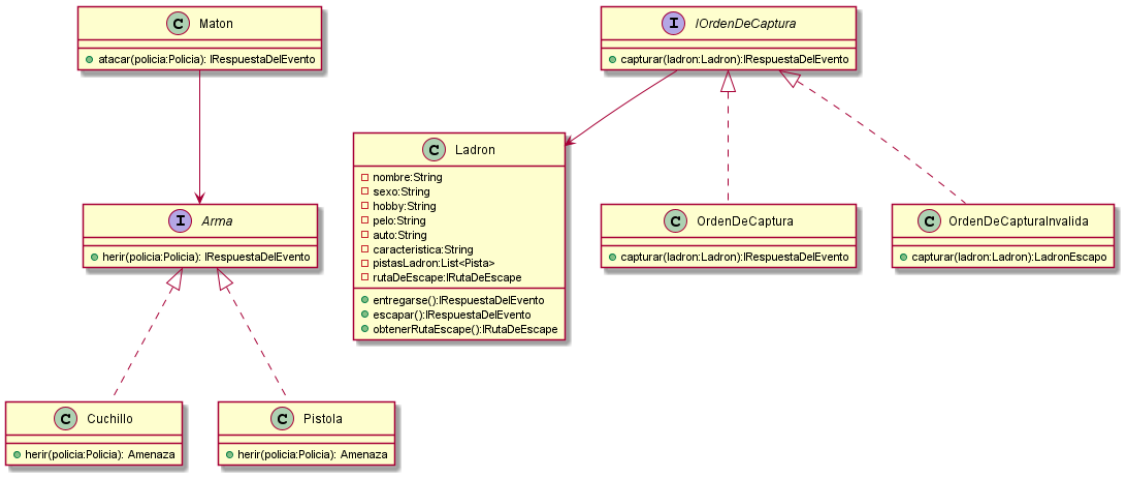

Figura 4:
Maton podra atacar al policia y retrasarlo; mientras que el ladron podra ser capturado, o no, dependiendo si el policia tiene una orden de captura.

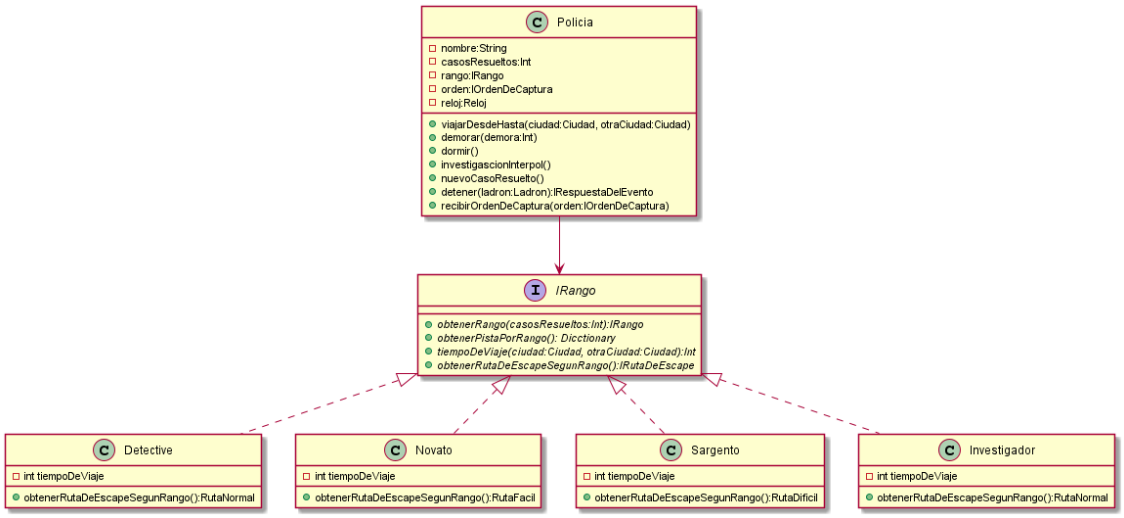

Figura 5:
Policia cuenta con cuatro tipos de rango, los cuales podra ir escalando a medida que cumpla las misiones de capturas.

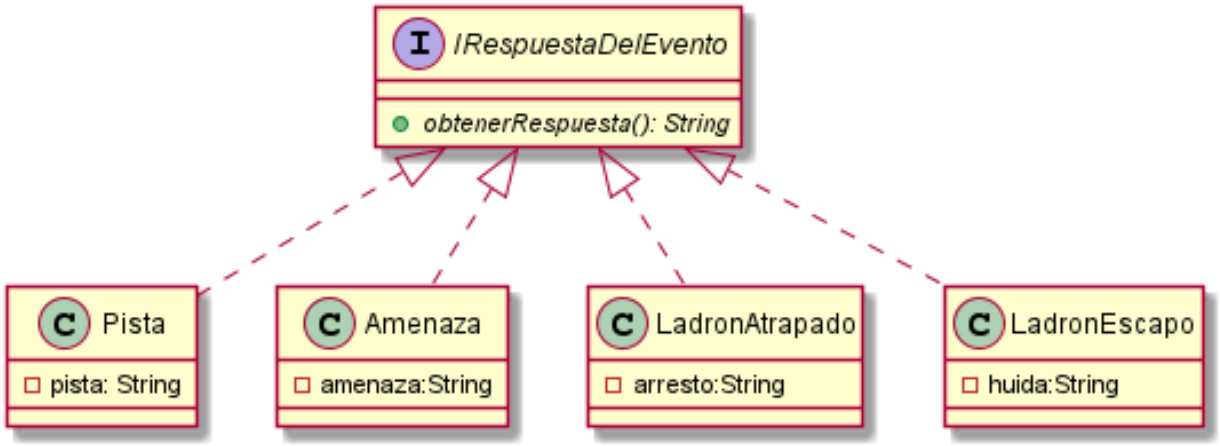

Figura 6:
hay distintos tipos de evento dependiendo de la ciudad y los edificios visitados

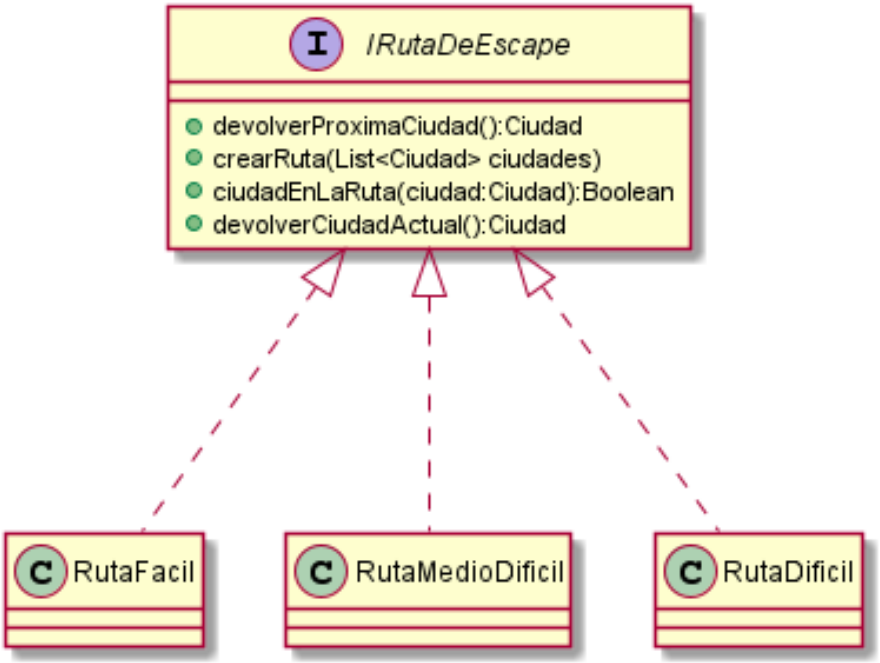

Figura 7:
hay 3 tipos de ruta de escape dependiendo de la dificultad del juego. Dependiendo cada una varia la cantidad de paises que visita el ladron. El metodo es en realidad devolverProximaCiudad(), no cantPaises()

3. Diagramas de secuencia

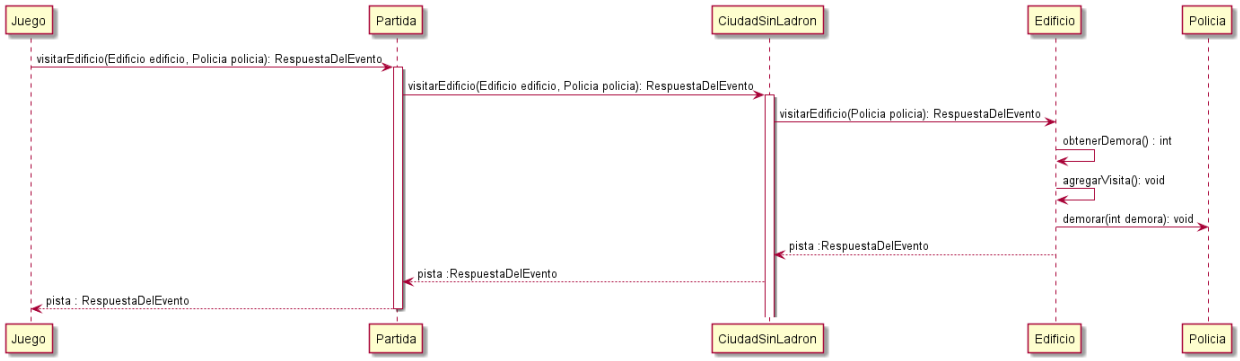

Figura 8:
Policia entra por primera vez a un edificio

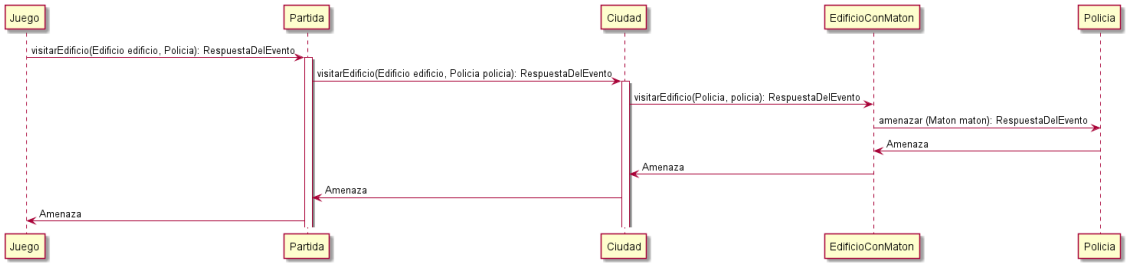

Figura 9:
Policia visita por primera vez a un edificio sin ladron en ciudad con ladron

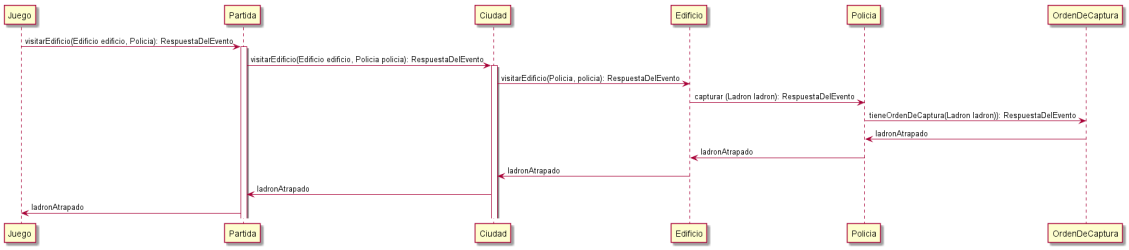

Figura 10:
Policia visita por primera vez un edificio y esta el ladron

4. Detalles de implementación

A continuación se mencionan los puntos más conflictivos del trabajo práctico y las estrategias utilizadas para resolverlos:

4.1. Objetivo General

Cuando se inicia el juego, el jugador comienza a las 9 am del lunes y debe atrapar al ladron antes de las 5 pm del domingo.

Cuando el jugador visita a algún edificio, utilizamos la interfaz IRespuestaDelEvento que permite devolverle al policia una Pista, un ataque del matón (si se encuentra en el edificio) o si el ladron se encuentra allí.

El matón que se encuentra en EdificioConMaton, se inicializa siempre con una instancia de Cuchillo como arma para atacar.

La clase BuscadorLadrones se encarga de cargar y guardar los datos del ladron que el jugador va recopilando en la computadora, para poder después buscarlos por filtro y emitir la orden de captura.

En caso de que el jugador encuentre al ladrón y lo intente capturar sin orden de captura, se va a devolver una nueva instancia de la clase LadronEscapo, que contiene un string que muestra lo ocurrido.

Utilizamos clases inicializadoras de ladrones, ciudades, y de pistas, que toman los datos de los archivos json que contienen la información necesaria sobre lo mencionado anteriormente.

Esta la interfaz RutaDeEscape que le devuelve al ladron la proxima ciudad que tiene que visitar . Del viaje se encarga la clase partida.

4.2. Principios de Diseño

Para este trabajo práctico los principios de diseño utilizados corresponden a los principios SOLID:

1) El principio de responsabilidad única. Cada una de las clases de nuestro modelo cumple una función única, sencilla y concreta. Este principio se cumple en todo el modelo pero se pueden mencionar algunos ejemplos específicos. La clase Pistola encargada de herir al policia y retrasarlo; o la clase Reloj que contara el tiempo que se lleva jugando.

2) El principio de abierto / cerrado. Las clases deben estar abiertas para la extensión pero cerradas para la modificación. Para aplicar este principio hacemos uso de interfaces que nos permiten agregar comportamientos nuevos si así se requiriese. Una de estas interfaces es, por ejemplo, Edificio, del cual existen tres tipos de edificios que lo implementan: con maton, con ladron y edificio con pista; y a su vez este ultimo podra ser un banco, un puerto, aeropuerto o biblioteca. Si se requiere agregar otro tipo de edificio solo basta con implementar la interfaz, sin tener que modificar el codigo original.

3) Principio de sustitución de Liskov. En nuestro programa se hace uso de ordenes de capturas. Pueden ser de dos tipos, valida o no valida y ambas implementan la interfaz IOrdenDeCaptura. Policia guarda en su estado una orden captura, pero no sabe si es valida o no; de hecho hasta puede cambiar en tiempo de ejecucion. Para evitar preguntar que tipo de orden es la que tiene para asi aplicar el funcionamiento adecuado cuando se encuentre con el ladron, es que el policia guardara la interfaz y no la implementacion concreta.

4) Principio de segregación de la interfaz. Cada clase va a implementar la interfaz que va a utilizar. En nuestro modelo, hay interfaces: IRango (con los distintos rangos que puede tener un policia, empezando con novato), IOrden de captura (orden valida o no valida) e IRespuesta del evento (donde se implementan diferentes eventos que pueden ocurrir mientras se juega). Cada una de estas interfaces solo posee unos pocos metodos para asegurar que todas las clases puedan implementarlos.

5) Principio de inversión de dependencias. Se depende de las abstracciones y no de las implementaciones. Por ejemplo, cuando el policia captura al ladron, policia no depende de las clases de orden de captura, ni ladron. Se podrian cambbiar ambas implementaciones sin comprometer el codigo de policia.

4.3. Patrón de Diseño: MVC

Con respecto al patrón de diseño, se utilizó Modelo-Vista-Controlador que divide el programa en tres grupos según las tareas que realiza.

Por un lado el Modelo, presenta las clases y objetos del dominio. La Vista corresponde a la interfaz, todo lo que se le muestra al usuario. Es una representación del estado del Modelo en un momento determinado. Y el Controlador actúa como intermediario entre el usuario y el modelo. Capta los cambios en la vista y los traslada al modelo y viceversa. Elegimos este patrón de diseño para nuestro TP porque separar según estas funciones es una forma muy ordenada y conveniente para realizar un software completo.

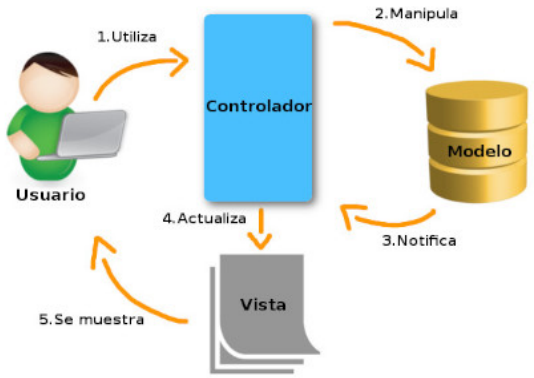

Figura 11:
Esquema del patrón de diseño MVC utilzado en nuestro juego Algothief.