FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Algo-thief

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2021

Alumno	Número de padrón	Email
Mauro Villagra	105241	mvillagra@fi.uba.ar
Esteban Frascarelli	105965	efrascarelli@fi.uba.ar
Agustin Ezequiel Sanchez Decouflet	107092	asanchezd@fi.uba.ar
Claudia Ramos	104596	cramos@fi.uba.ar
Camila General	105552	cgeneral@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo practico de la materia de Algoritmos y Programación ||| que consiste en desarrollar una aplicación relacionada al juego de Carmen Sandiego. Nuestro objetivo es aplicar los conceptos explicados a lo largo de la materia, utilizando un lenguaje de tipado estático llamado Java. Primordialmente, haciendo énfasis, en el desarrollo de un modelo orientado a objetos.

2. Supuestos

2.1. Desde cada ciudad, se muestran siempre las mismas ciudades en el mapa

Siempre que el policia viaje a una ciudad, desde esa ciudad se le van a mostrar en el mapa las mismas tres posibles ciudades para viajar y asi con cada ciudad (podrian llegar a repetirse).

2.2. Se puede mostrar mas de una pista del sospechoso en la misma ciudad

Se va a poder dar mas de una pista del sospechoso en la misma ciudad, pero cada pista (que describe una caracteristica del ladron) solo se va a mostrar una vez.

2.3. Se pueden emitir mas de una orden de arresto

Una vez emitida una orden de arresto, pueden cambiarse las caracteristicas del sospechoso ingresadas y si queda un unico sospechoso distinto al de la primer orden de arresto, se va a emitir una nueva orden y esta va a reemplazar a la anterior.

3. Diagramas de clase

Un diagrama de clase es un diagrama estático en el cual se representa la estructura de un sistema compuesto por clases, reflejando así sus atributos, métodos y las relaciones con otros objetos. A continuación se presentan algunos diagramas de clase correspondientes al trabajo practico.

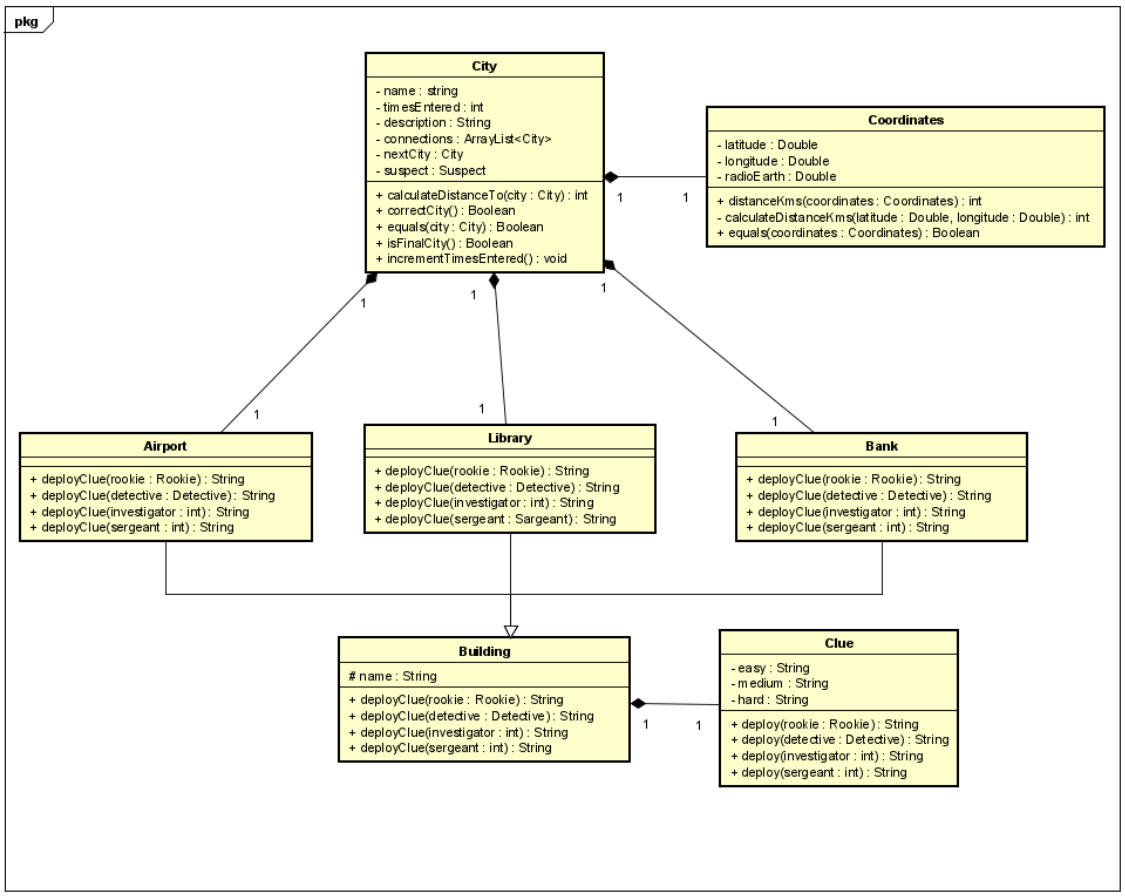

Figura 1: Relación emtre la ciudad y los diferentes edificios.

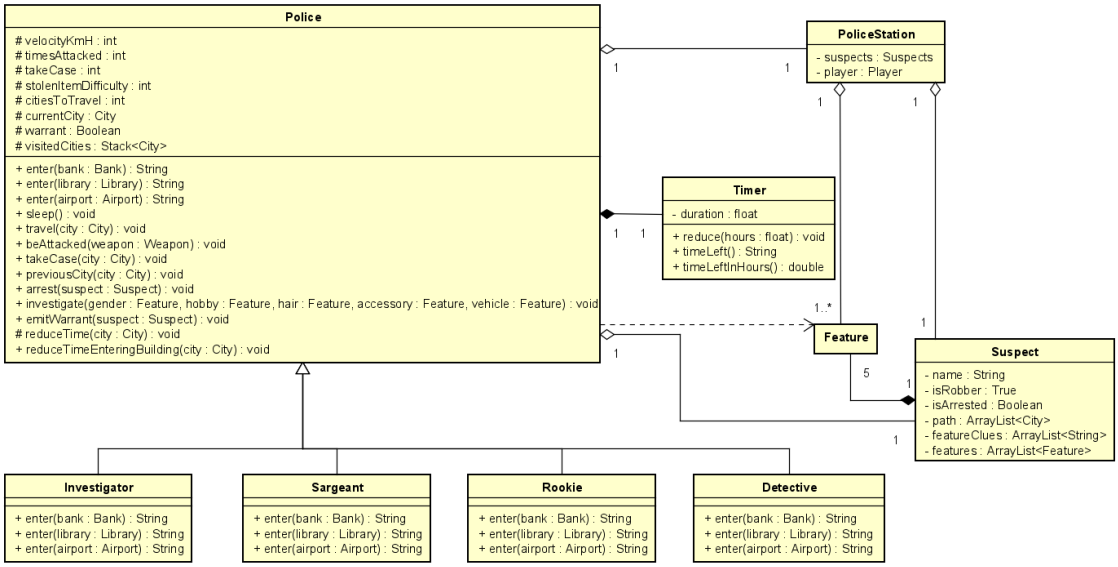

Figura 2: Relación entre la clase Police con los diferentes tipos de Police y la clase PoliceStation.

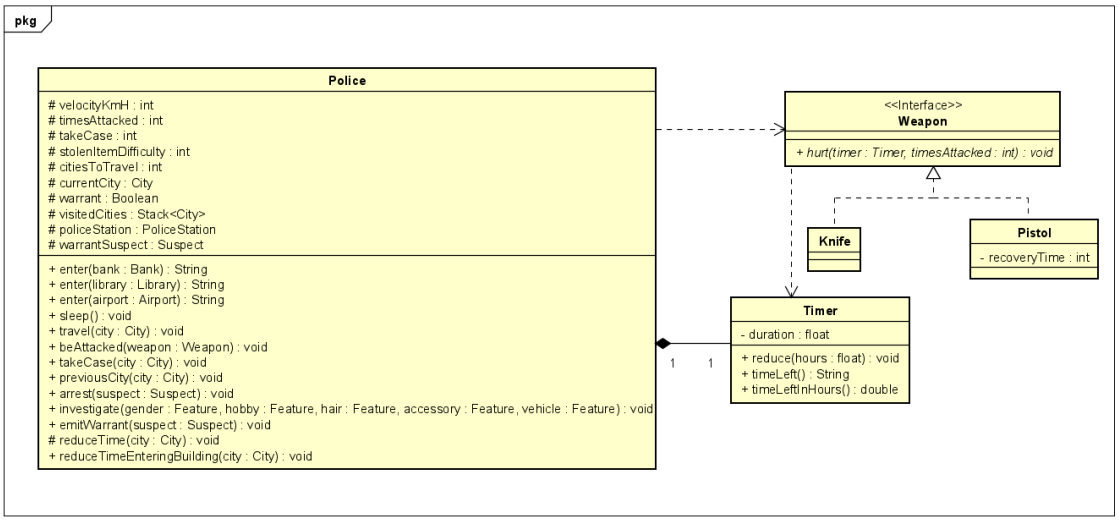

Figura 3: Relación entre la clase Police y las clases que implementan la interfaz Weapon.

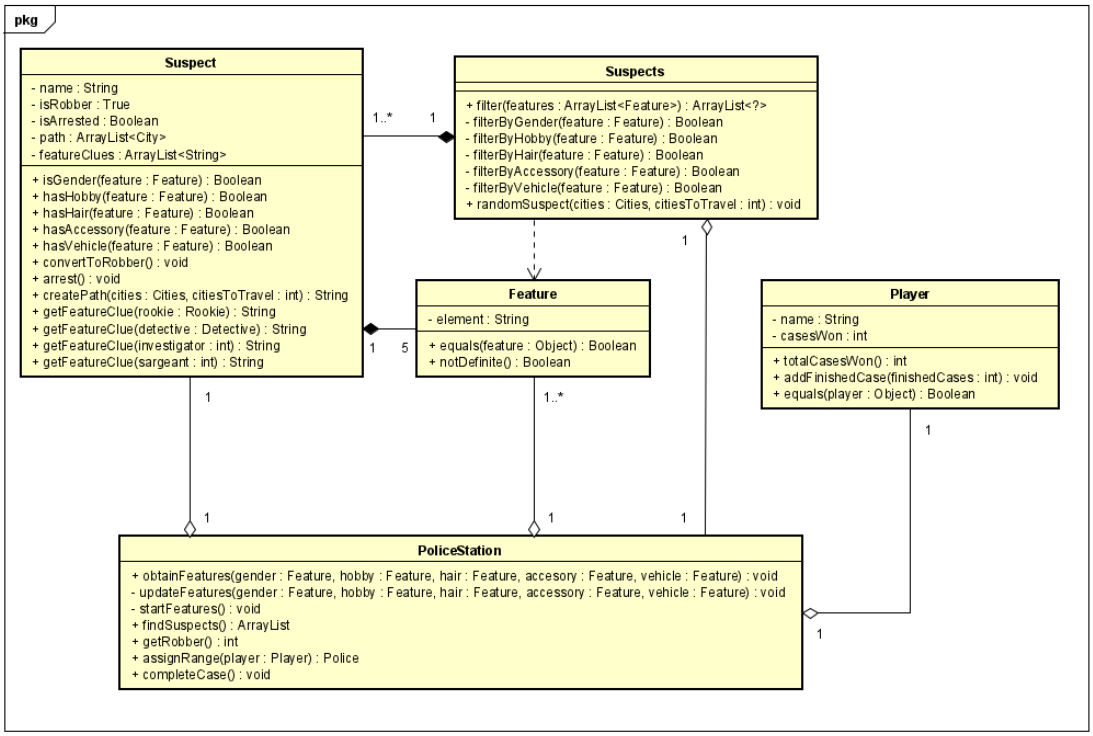

Figura 4: Relación entre las clases PoliceStation, Suspect y Suspects .

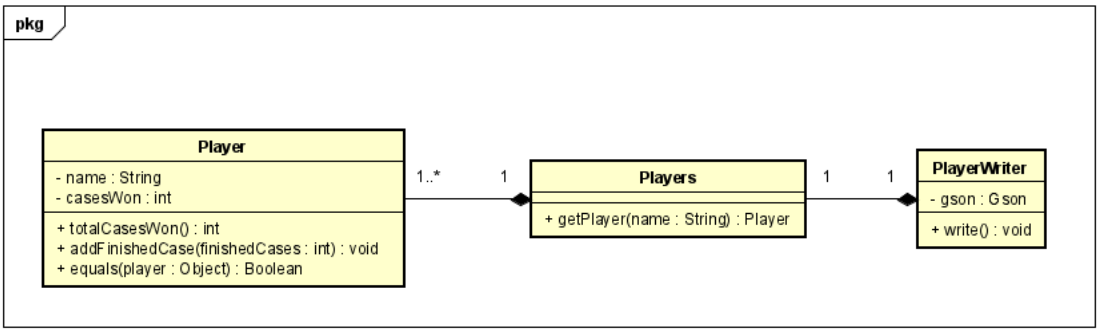

Figura 5: Relación entre las clases Player, Players y PlayerWriter.

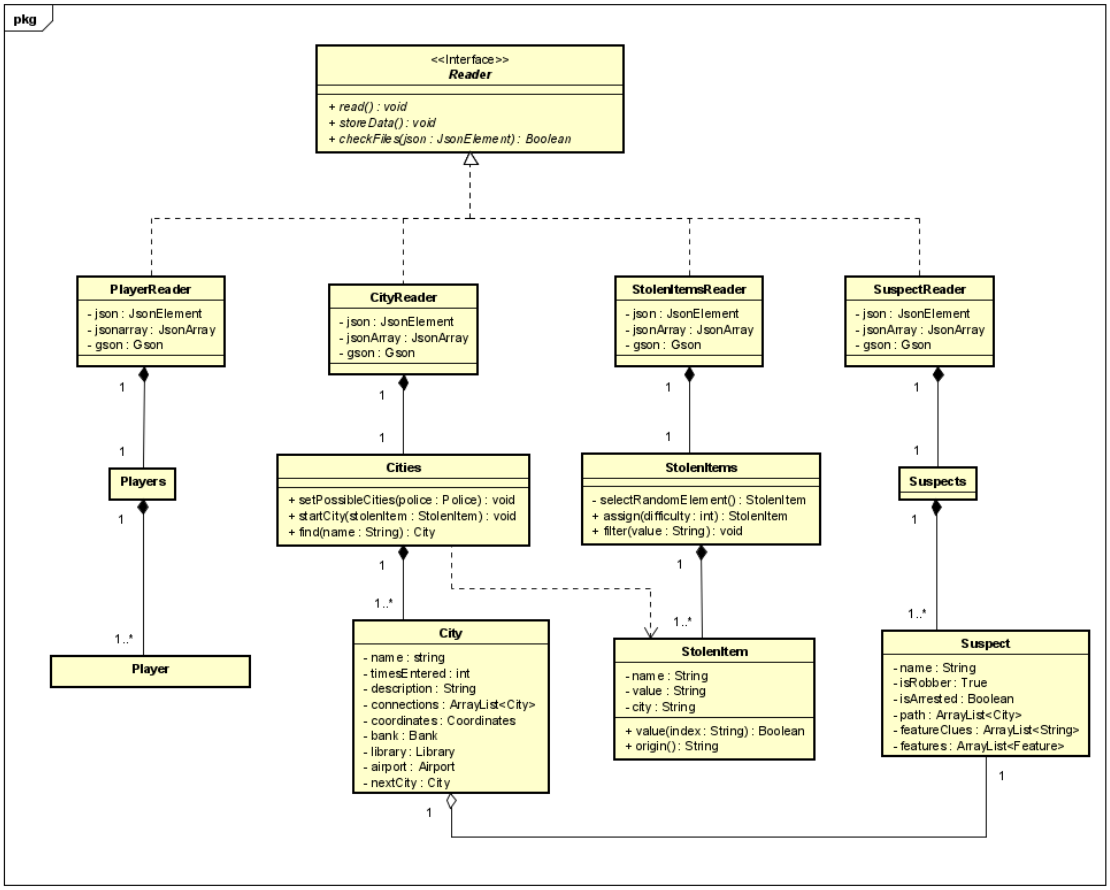

Figura 6: Relación entre los lectores y las diferentes listas.

4. Diagramas de secuencia

Un diagrama de secuencia muestra la interaccion entre un conjunto de entidades en un determinado caso de uso a lo largo del tiempo. A continuacion se presentaran diferentes diagramas que mostran diferentes situaciones dadas dentro del modelo de dominio:

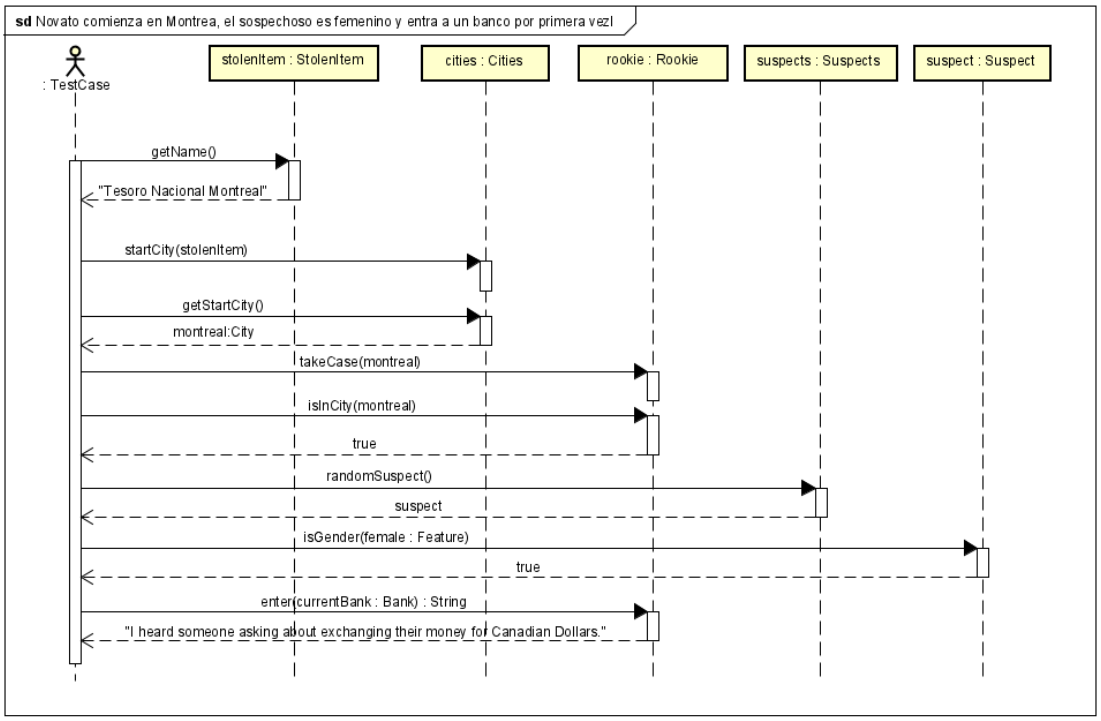

Figura 7: Novato comienza en Montrea, el sospechoso es femenino y entra a un banco por primera vez.

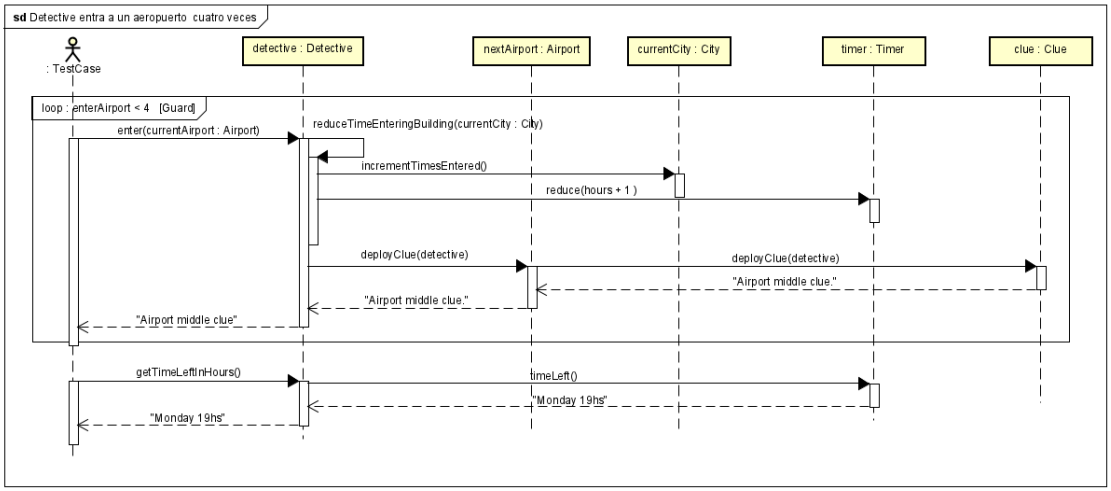

Figura 8: Detective entra a un aeropuerto cuatro veces.

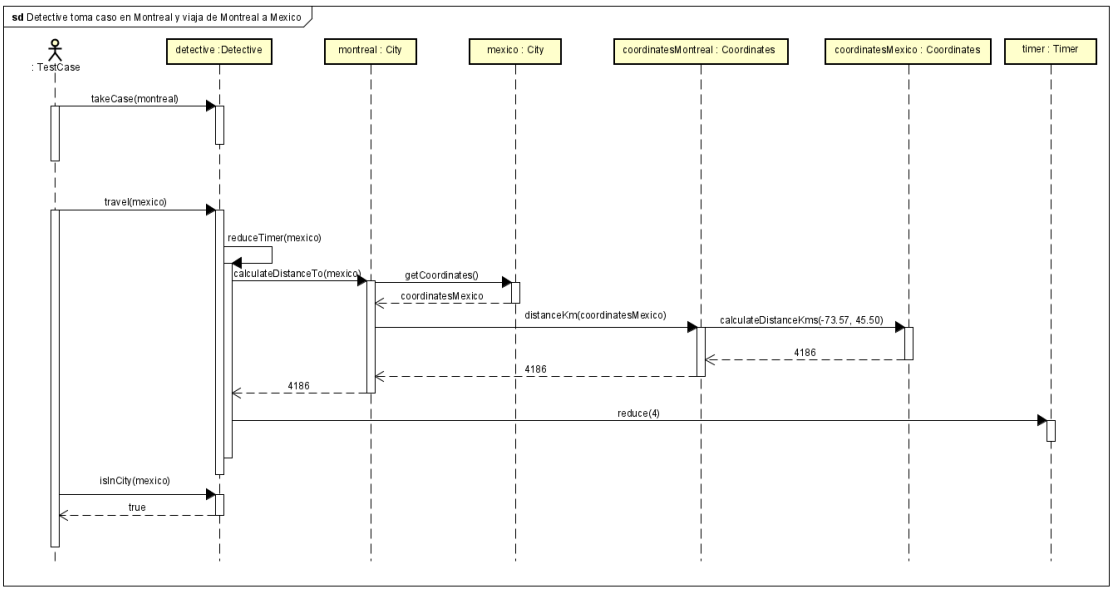

Figura 9: Detective toma caso en Montreal y viaja de Montreal a Mexico.

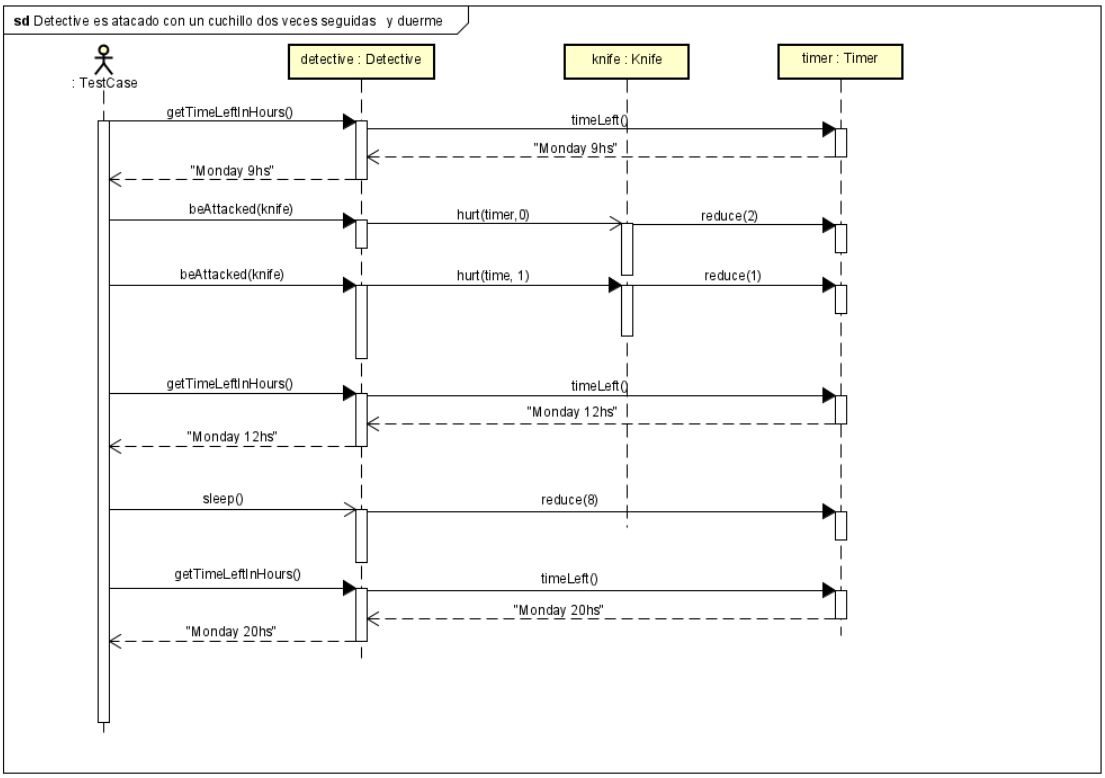

Figura 10: Detective es atacado con un cuchillo dos veces seguidas y duerme .

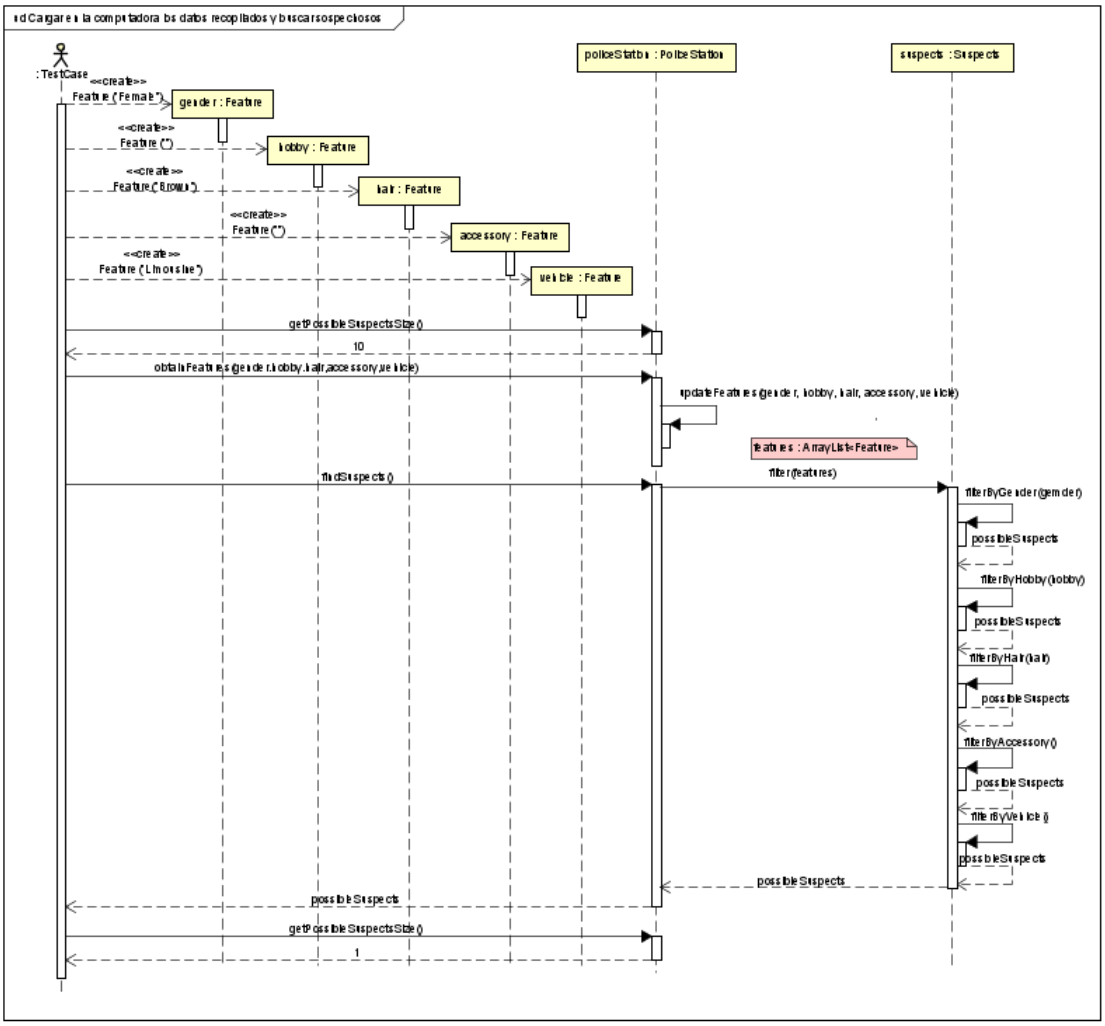

Figura 11: Cargar en la computadora los datos recopilados y buscar sospechososr.

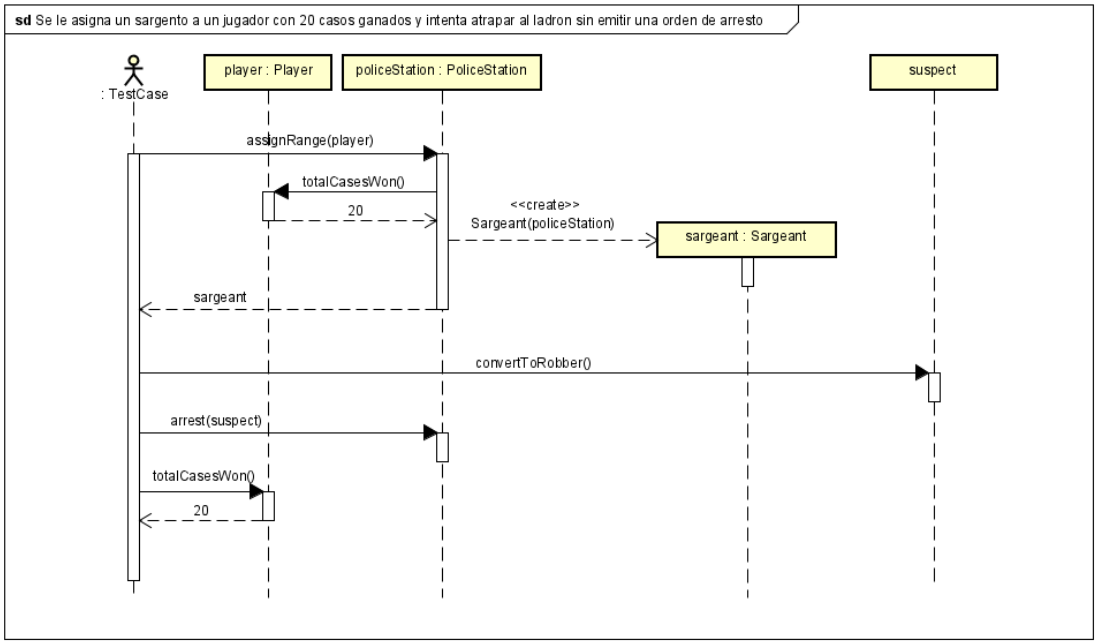

Figura 12: Se le asigna un sargento a un jugador con 20 casos ganados y intenta atrapar al ladron sin emitir una orden de arresto.

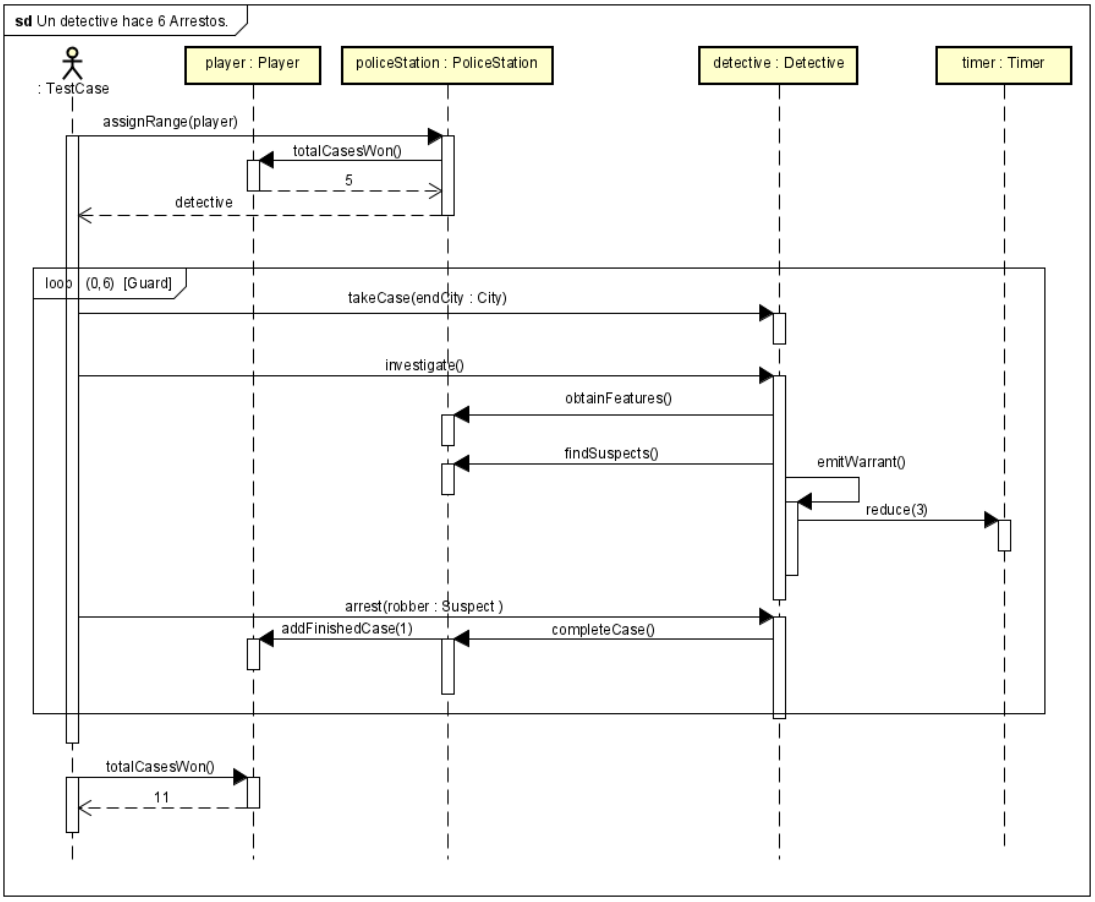

Figura 13: Un detective hace 6 Arrestos.

5. Diagramas de paquete

El diagrama de paquete se emplean para mostrar la organización y disposición de diversos elementos de un modelo en forma de paquetes. Por ejemplo, un paquete podría agrupar clases, pero también objetos o casos de uso, e incluso otros paquetes. A continuación se presentan algunos diagramas de paquete correspondientes al trabajo:

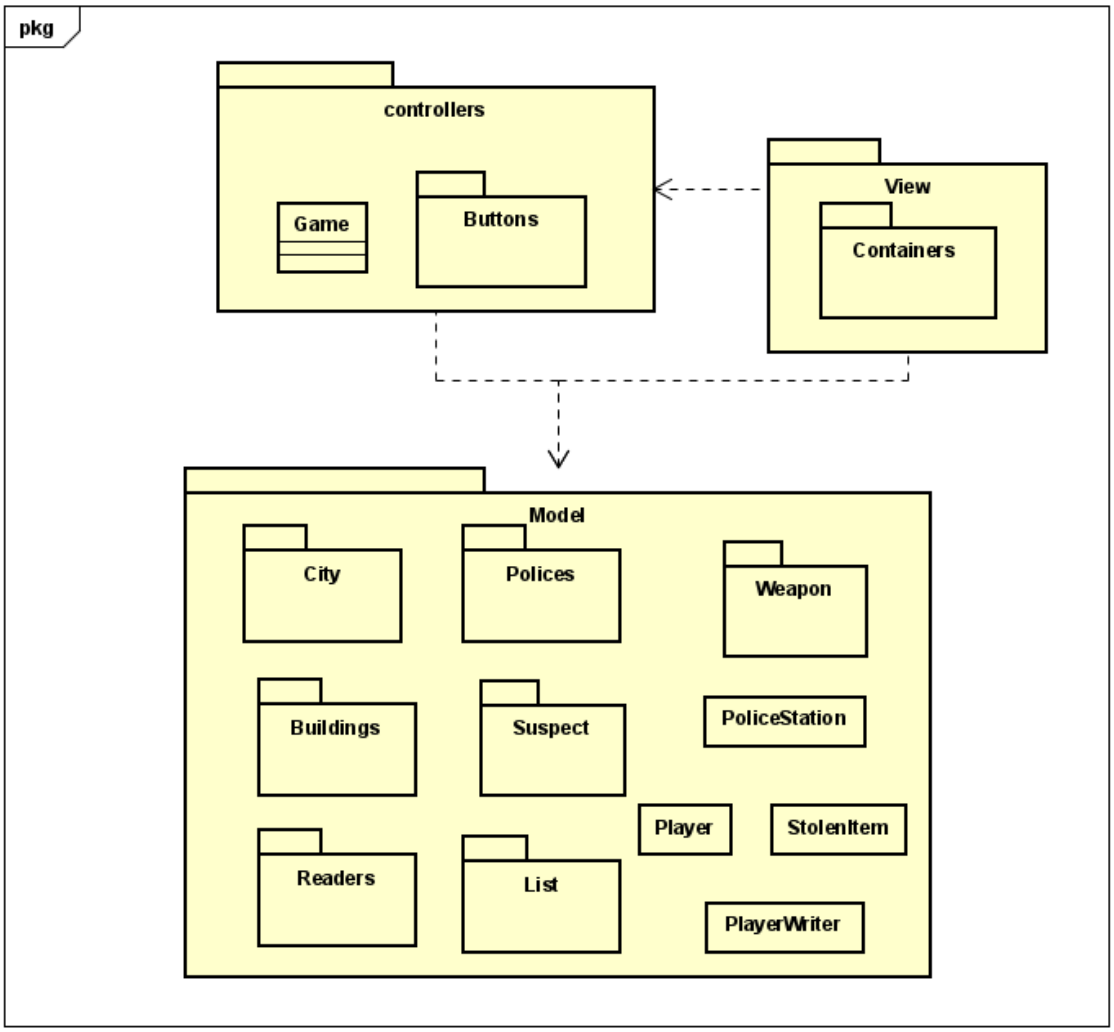

Figura 14: Diagrama de paquete general utilzando el patrón MVC.

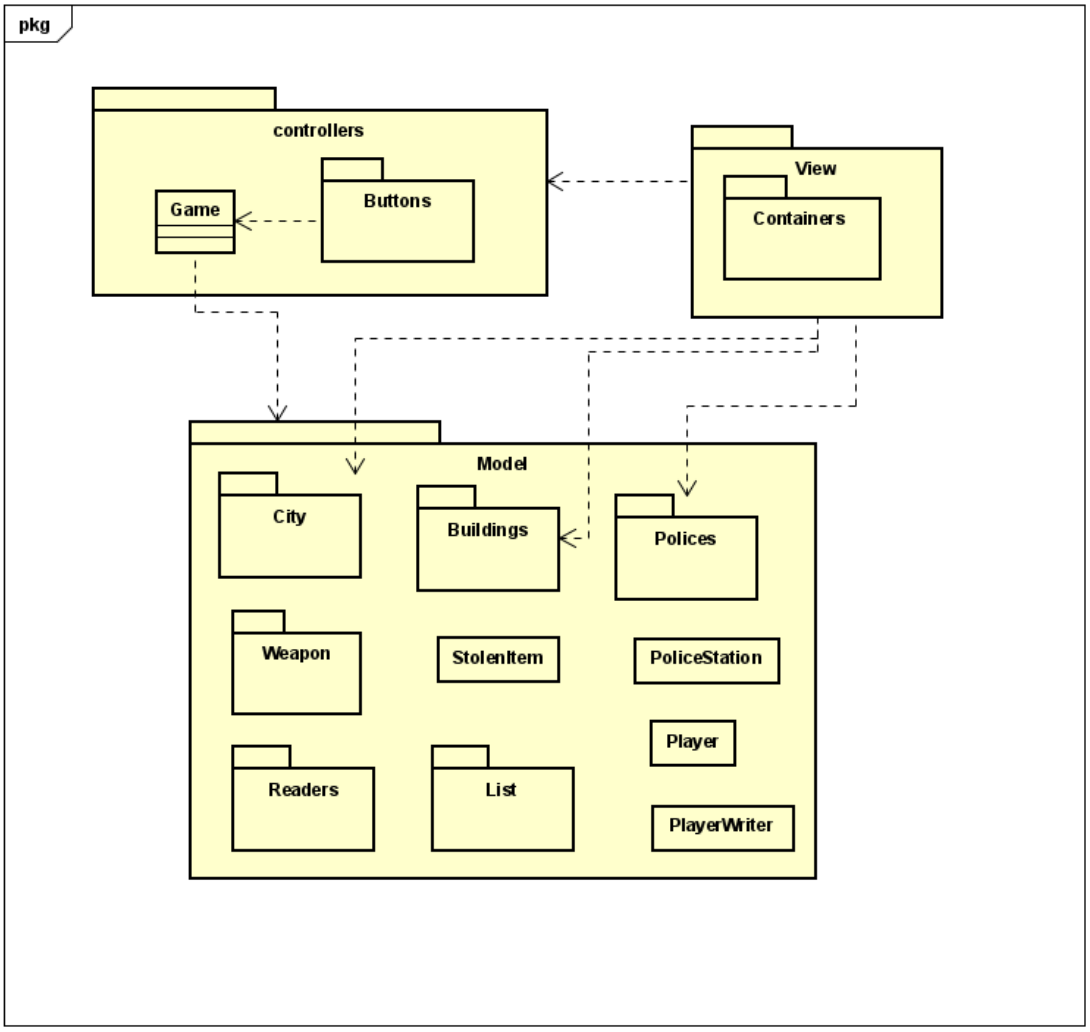

Figura 15: Diagrama de paquete mostrando las relaciones entre las diferentes clases de los diferentes paquetes.

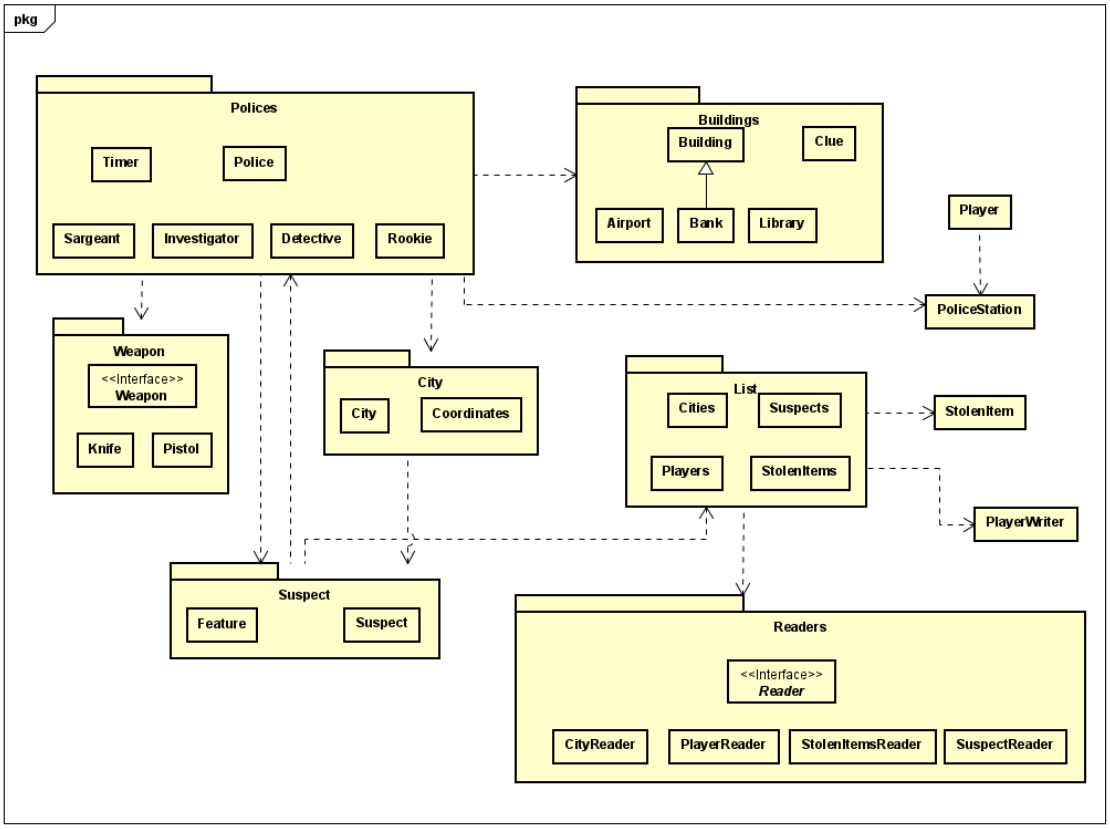

Figura 16: Diagrama de paquete en detalle del paquete Model.

6. Diagrama de estado

Los diagramas de estado muestran el conjunto de estados por los cuales pasa un objeto pero tambien puede mostrar el comportamiento general de un conjuntos de estados. A continuación se muestra le diagrama de estado que plasma el comportamiento general del modelo:

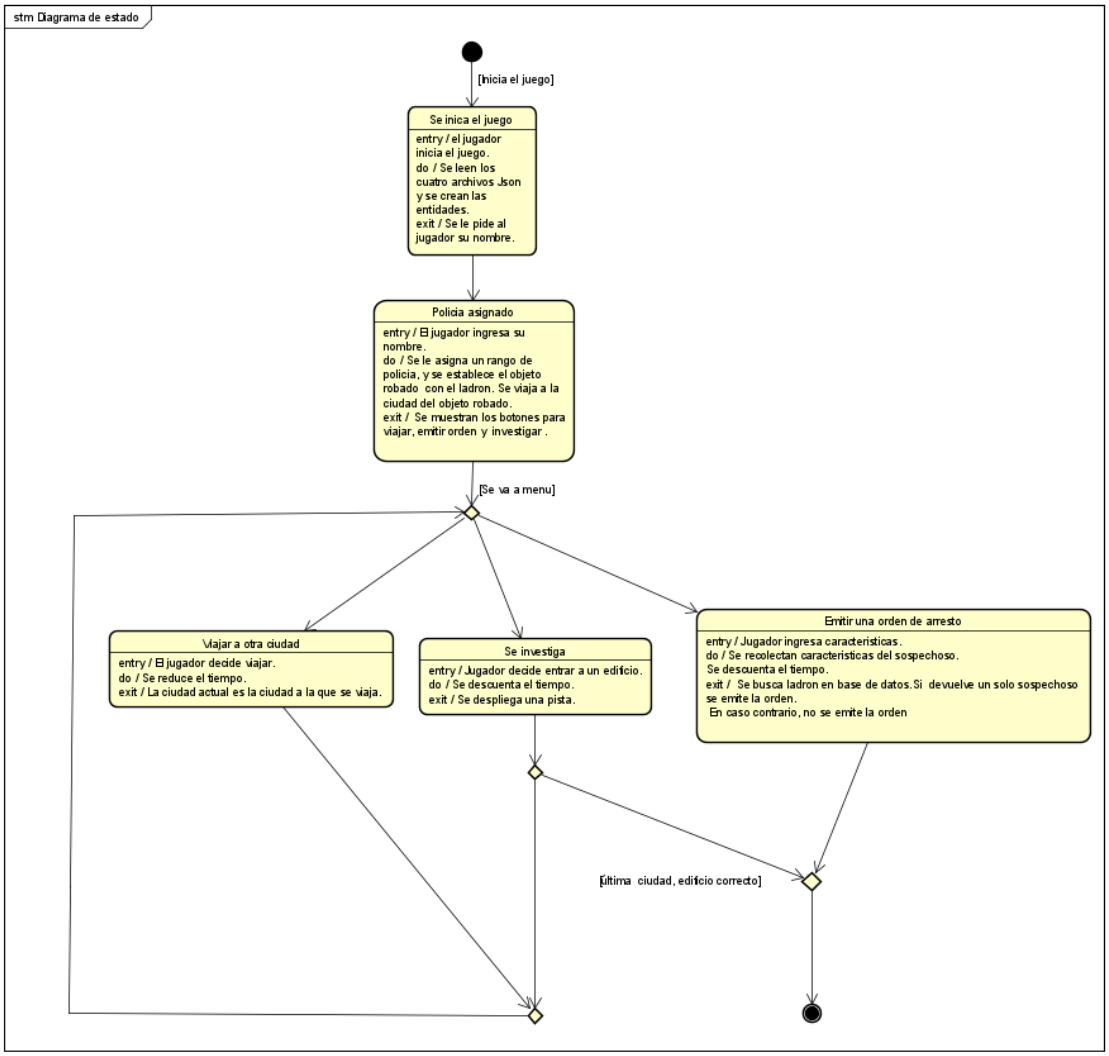

Figura 17: Diagrama de estado del juego en general.

7. Detalles de implementación

7.1. Uso de herencia para los rangos del Policia

En el caso del policia optamos por darle una identidad a cada rango del mismo, el cual dependia del total de casos ganados del jugador. Elegimos implementar una clase madre abstracta llamada Policia; la cual hereda a sus hijos novato, detective, investigador y sargento, porque nos dimos cuenta de que además de cumplir la regla: 'es un', respetaban el contrato de su clase madre casi en su totalidad salvo por tres metodos abstractos que los relacionaban con los edificios a los que iban a entrar, y dependiendo del rango(tipo de objeto) del policia que ingresaba se enviaban distintos mensajes a los mismos (como se explica en el siguiente punto), haciendo uso de polimorfismo en esta tarea.
Al ver que las entidades de los rangos del Policia solo se diferenciaban en el comportamiento que marcamos antes, vimos factible el aplicar herencia, sacrificando la escalabilidad de las clases hijas sabiendo que no pueden tener mucho mas comportamiento que la clase madre (ya que se violaria el principio de sustitucion), en provecho de no repetir todo el codigo que comparten el Policia y

sus clases hijas (respetando el principio DRY).

7.2. Uso de Double Dispatch para el problema de las pistas

Uno de los puntos mas conflictivos del TP lo tuvimos en la implementacion de una solucion a la problemática de tener diferentes tipos de pista, que dependian del edifico en el que estaban, y a su vez del rango del Policia el cual ingresaba al edificio.
Por esto detectamos que el metodo que iba a desplegar la pista no solo dependia del receptor, que en nuestro caso era el edificio al que se ingresaba, sino tambien del argumento que se le pasaba por parametro el cual decidimos que iba a ser el rango del policia ("notificandonos"la dificultad de la pista a ser desplegada, basandonos en el tipo de objeto), siendo cada uno de ellos una clase concreta que heredaban de una clase abstracta que nos permitió aplicar polimorfismo a la hora de elegir la pista que se iba a desplegar, como explicamos en el punto anterior, clave para resolver el problema. Justo por estas razones decidimos aplicar Double Dispatch, para elegir correctamente la pista a devolver tomando en cuenta el rango del policia y el edificio al cual haya entrado.

7.3. Uso del patrón Fachada

En nuestro modelo utilizamos el patrón Fachada al crear las clases Cities, Suspects, StolenItems y Players, debido a que se buscaba implementar clases que tengan más comportamiento que un ArrayList. Por ejemplo, en la clase Suspects, se le agrega el comportamiento de que pueda filtrar de acuerdo a diferentes clases Feature. También como la clase Cities que puede devolver, de acuerdo a un rango de policía,una cierta cantidad de ciudades aleatorias. Por otra parte, en la clase StolenItems puede devolver un objeto robado de acuerdo a una dificultad. Finalmente, en la clase Player, el comportamiento agregado es que al devolver un jugador, si no lo encuentra, devuelve un jugador nuevo.

Por otro lado, se utiliza el patrón Fachada en las clases CityReader, StolemItemReader, PlayerReader y SuspectReader. Estas clases, implementan el metodo read de la interfaz Reader para leer los archivos Json con ayuda de la libreria Gson encapsulando en ellas los metodos que les proporciona la misma. De manera que las clases que se encargan exclusivamente de leer los archivos no interactúan directamente con el modelo y, de esa manera, poder bajar la dependencia entre el modelo y la lectura de un archivo.

7.4. Uso de polimorfismo implementando una Interfaz

Las clases CityReader, StolenItemReader, PlayerReader y SuspectReader usan la interfaz Reader ya que implementan la misma firma pero cada uno guarda la informacion leida del Json, en sus respectivas listas(Cities, StolenItems, Players y Suspect).
También las clases Pistol y Knife implementan la interfaz Weapon. Las dos clases entienden el método herir pero el comportamiento de cada arma es distinta. Pistol y Knife descuentan el tiempo de forma distinta. Si bien cada clase cumple la relación: .es un", en un futuro puede suceder que en el caso de los lectores se necesite utilizar un atributo o método adicional que en los demás lectores no se desea. Así como también en el caso de una nueva arma, podría necesitar de otras entidades distintas a las de Pistol y Knife, a la hora de herir. Por lo que la interfaz nos ayuda a posponer estas decisiones de implementación.

7.5. Uso del patrón MVC

Utilizamos el patrón MVC para organizar mejor las dependencias entre el modelo y la interfaz. Debido a que nos favoreció en poder tener tanto el modelo y el diseño de la interfaz menos acoplados mediante un intermediario, en nuestro caso la clase Game. De manera que, al realizar diferentes cambios, no repercute de manera drástica tanto en el modelo como la interfaz.

8. Excepciones

FinalCityException Si bien no es una excepcion como tal ya que se lanza cuando el policia llega a la ultima ciudad, se usa de esta manera porque es la forma mas sencilla para que el final del juego no tenga un codigo muy intrincado y se llegue a la solucion de una manera mas directa. Ademas optamos por tomarlo como una excepcion ya que no nos parecia equivoca la idea de que este era un evento particular.