Programación con Objetos 1

TP Final – Carmen Sandiego

Barria – Carrevedo

10/12/2020

Informe sobre desarrollo, alcance y conclusiones del trabajo realizado.

Programación con Objetos 1

TP Final – Carmen Sandiego

Alcance
Mediante la utilización de Smalltalk, dentro del entorno de Cuis University, se desarrolla el equivalente a una partida del videojuego Carmen Sandiego. El proyecto se realiza basado en el paradigma de la programación con objetos, haciendo uso de elementos como clases, mensajes, colecciones; respetando conceptos como herencia, encapsulamiento y polimorfismo.

Implementaciones logradas
Las implementaciones solicitadas en el enunciado fueron implementadas en su totalidad. Un detective puede: consultar una lista de países, viajar a ellos, consultar una lista de sospechosos, interrogar testigos, obtener pistas que no se repiten, filtrar en base a esas pistas y emitir una orden de arresto. Siempre consumiendo y controlando el tiempo restante de la partida. En base a correcciones, se pudo mejorar incluso algunos inconvenientes de encapsulamiento.

Implementaciones pendientes
El proyecto se entrega con un refactor pendiente del cual el equipo es consciente. Todas las responsabilidades se armaron sobre el detective, cuando debería haber sido sobre un objeto que representara a la partida. No fue posible trasladarlo y reacomodar las implementaciones. La idea inicial fue la correcta, incluso cuando se representó gráficamente la idea original del proyecto (gráfico en página 2), se ideó de esa forma. Pero en el armado se desvió el camino.

Características extras
Se habían implementado unas pequeñas modificaciones que nos parecían acertadas, pero se debieron sacar en base a la corrección de la pre entrega. Levantar una excepción cuando el detective no tenía tiempo suficiente para viajar o interrogar. Nos parecía apropiado que no lo hiciera y pudiera disponer del tiempo para intentar emitir una orden de arresto (se agregan en un apartado al final las implementaciones que se borraron).

Historia del Carmen Sandiego

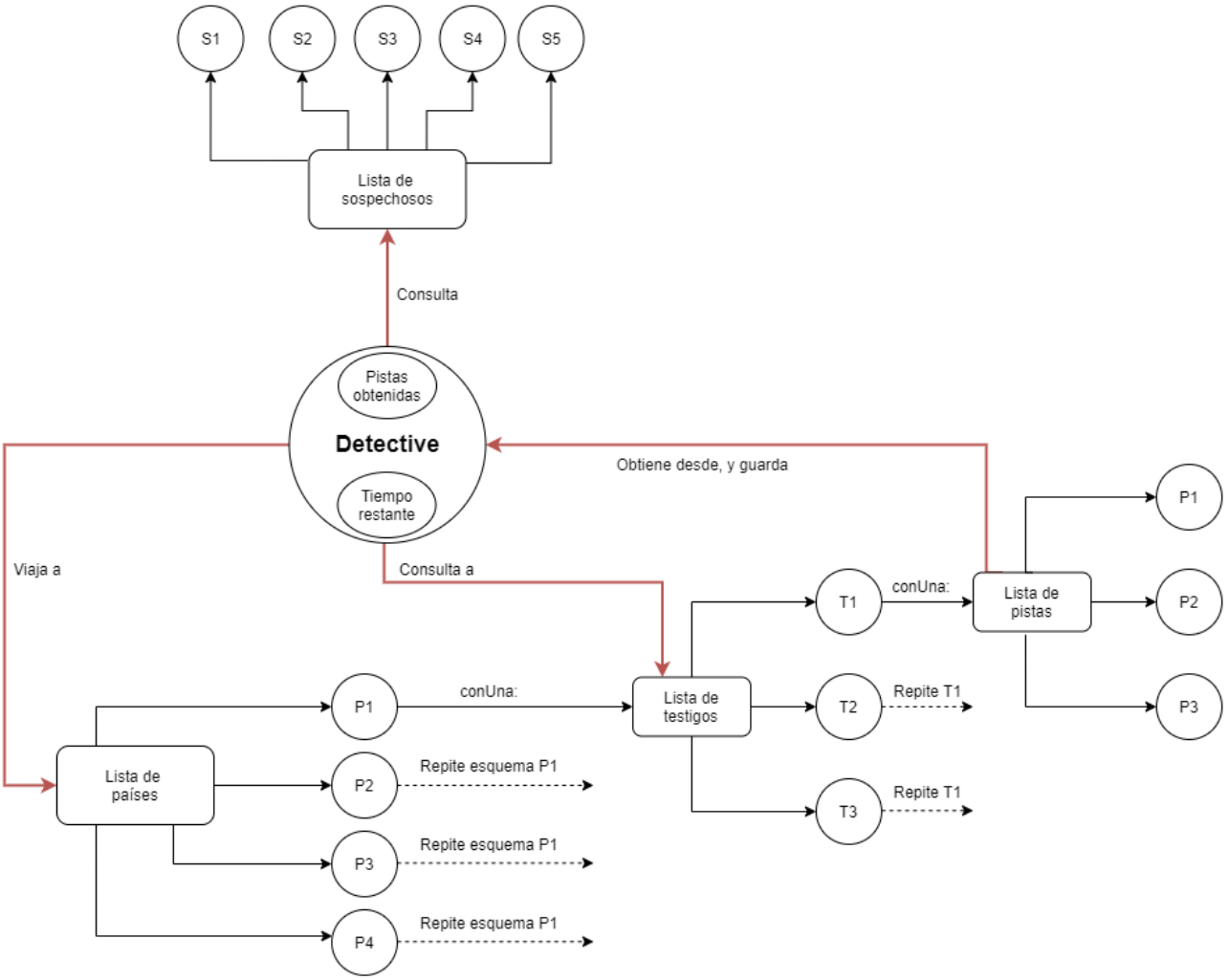
El videojuego es una aventura gráfica cuya antagonista es Carmen Sandiego, una ladrona de guante blanco que lidera la organización criminal V.I.L.E. . Está especializada en robos a gran escala de bienes históricos.

El jugador forma parte de una agencia de detectives, deberá recabar pistas y viajar a otros lugares para encontrar al ladrón antes de una fecha límite.

Las pistas son dadas por los testigos, y permiten encontrar al ladrón en base a un razonamiento lógico.

La detención se produce mediante una orden de arresto, que se consigue al obtener un único sospechoso que coincida con las pistas conseguidas.

Modelo



Este esquema muestra las principales comunicaciones entre objetos.

Tenemos al detective, quien puede consultar una lista de 5 sospechosos. También puede viajar a un destino eligiendo de una lista de 4 países, donde cada país posee una lista de 3 testigos, quienes a su vez poseen una lista de 3 pistas cada uno. Las pistas constarán de destinos y características físicas, que serán guardadas en un colaborador interno del detective, con la cualidad de no guardar pistas repetidas. Otro colaborador interno irá registrando la cantidad de horas que quedan para resolver el caso.

Progreso

- En un primer paso, se armó en un borrador una lista con los objetos más generales que podíamos identificar a través del enunciado, anotando también, en algunos casos, los mensajes que eran evidentes durante la lectura.
- Se comienza a realizar TDD, trabajando sobre la ubicación inicial del detective. Viajes y cambios de ubicación + consumo de tiempo.
- El punto anterior obligó a la creación de países y de un proveedor de distintos tipos de viaje.
- Se crean los sospechosos, testigos y detective, generando una clase abstracta Personaje. Ésta última fue borrada por una recomendación en una corrección.
- Se intentó hacer uso del objeto Factory para la creación de listas, pero luego se eliminó.
- Se modelan las pistas e interrogatorios.
- Se realiza el filtrado en base a las pistas.
- Se agregan excepciones y validaciones.
- Se crean la orden de arresto, imprimible en un archivo de texto.

Dificultades

- La primera dificultad fue conceptual. Pensar el juego para una única "partida" o para varias. En caso de que fuera para una, las ubicaciones, pistas, y otras características podrían estar establecidas en un setup. Pero si tuviera que ser para distintas partidas, se deberían utilizar distintos tipos de mensajes de clase, para poder dar cierta aleatoriedad.
- Representar cosas como objetos distintos innecesariamente. En un principio, cada país era un objeto distinto, lo mismo pasaba con cada pista u objeto robado. Luego notamos que no existía comportamiento distinto entre cada uno, lo que nos llevó a crear un único objeto con distintos colaboradores. Esto obligó a un refactor grande.
- Establecer países limítrofes.
- El manejo de las pistas. Primero eran distintos objetos. Luego estuvieron divididas en 2 tipos, pistasGeograficas y caracteristicasFisicas, con la idea de poder diferenciar qué pista nos decía dónde ir. Esto también se intentó resolver con un diccionario. Pero finalmente nos dimos cuenta que manejándolas únicamente como strings era la forma correcta, ya que el detective (usuario) sólo necesita consultar la lista de pistas para 'leerlas' y en base a eso decidir. Incluso por eso también es posible viajar a un país equivocado, no debe ser una decisión del sistema.
- El manejo de respuestas booleanas. En varios casos fueron manejadas con respuestas con strings.
- **Aclaración importante: el test número 12, de los tests "ModeloDePartida", falla aleatoriamente. Pasa satisfactoriamente varias veces, y luego no pasa.**

Excepciones eliminadas

validarQueDelicuenteEstuvoEnElPaisDe: unTestigo

^ (((unTestigo ubicacion) esRutaDeEscape) = ('es ruta de escape')) ifFalse: [self error: self noPuedeRecibirPistasEnUnPaisDondeNoEstuvoElDelincuente].

La intención era no interrogar en un país donde el delincuente no estuvo, pero no era lo que pedía el enunciado.

validarQueHayTiempoSuficienteParaInterrogatorio

^ (self tiempoPorInterrogatorio < tiempo) ifFalse: [self error: self noTieneTiempoSuficienteParaRealizarInterrogatorio]

Nos parecía bueno avisar que no se podía interrogar, pero que la partida seguía para emitir la orden de arresto. Una similar había con viajaA: .

Conclusiones

Se mejoró en gran medida el uso en general de Smalltalk y la aplicación de TDD, en particular el ejercicio de establecer casos básicos que después deban dejar de funcionar. La creación de proveedores y excepciones, por repetición terminó saliendo de manera fluida. Gracias a algunas correcciones se comprendió más en profundidad el cuidado para no romper el encapsulamiento y no resolver cosas dentro de un mensaje, que podrían resolverse con otro mensaje.

A pesar de no haberlo podido refactorizar, entendemos que nuestro principal error es haber delegado toda la responsabilidad al detective. No es él quien debería saber cuánto tiempo de partida queda, por ejemplo. U objetos como sospechosos y países, deberían ser parte de una Partida que nucleara todos los objetos que intervienen.

Fuera de eso, estamos conformes con el funcionamiento y con lo logrado.