PONTIFICIA UNIVERSIDAD CATÓLICA ARGENTINA

PROGRAMACIÓN ORIENTADA A OBJETOS II

Carmen Sandiego en Smalltalk

Alumno: Lucas Ariel Saavedra
Profesor: Nahuel Garbezza

1 OBJETIVO

Realizar en Smalltalk el modelo del juego **Carmen Sandiego**: un detective debe adivinar quien fue el ladrón de un objeto preguntando a testigos para obtener pistas de quien puede ser ladrón o donde fue a partir de una lista de sospechosos. Si el detective se queda sin tiempo o se equivoca de sospechoso, pierde.

2 DESARROLLO

En el modelo, tenemos a los siguientes objetos:

- **Sospechosos (*Suspect*):** De cada uno de estos, se sabe el color de ojos, forma y color del pelo, color de tez, contextura física, sexo, y altura. Uno de los sospechosos es el **culpable**, mientras que los demás, obviamente, **no lo son**.
- **Testigos (*Witness*):** El testigo es el que le provee pistas al detective. Puede ser que los testigos no tengan ninguna pista por lo que el detective puede perder tiempo.
- **Países (*Country*):** Cada país tiene 3 testigos. Si el sospechoso paso por el país, los testigos tendrán pistas. En caso contrario, no saben nada (no dan ninguna pista). Los países pueden tener límites con otros.

- **Detectives (*Detective*):** Es el encargado de atrapar al sospechoso. Por cada interrogación que hace y por cada viaje, pierde una cierta cantidad de tiempo. Puede emitir una orden de arresto en cualquier momento.

Al arrancar el juego, el detective (que puede ser de la interpol, ser uno persuasivo o uno normal) arranca en un país que es donde el ladrón robo el objeto de gran importancia. Este país tiene testigos con pistas. El detective puede pedirles a los testigos pistas que va a ir guardando para ir sabiendo quien puede ser el sospechoso y a donde se podría haber ido. ¿Como le damos las pistas a los testigos? En un principio, los países tienen testigos sin pistas, pistas sobre el mismo país, y un nombre. Un ejemplo de creación de un país determinado es el siguiente:

```
argentina := Country withName: 'argentina'
                     andClues:#('tango' 'barbacue' 'empanada').
```

Si no le pasamos pistas a los testigos de Argentina, quedan como testigos "sin pistas". Ahora bien, si queremos pasarle pistas, esto queda cargo del objeto *argentina*:

```
argentina   giveWitnessesCountryCluesFrom: arrayOfCountriesWhereSuspectWas
            andSuspectCluesFrom: guiltySuspect.
```

Donde *arrayOfCountriesWhereSuspectWas* es un arreglo de países donde el ladrón pasó y *guiltySuspect* es el sospechoso que el detective tiene que encontrar.

Estando en un país, el detective puede preguntarles a los testigos pistas. Por cada pregunta que hace, el detective pierde una cierta cantidad de tiempo. Si el detective es persuasivo, por cada pregunta gana dos pistas. La forma en que el detective le pregunta a los testigos varía dependiendo de qué tipo de detective hablemos. Si hablamos de un detective normal o el de la interpol:

```
askTo: aWitness
        self validateWitness: aWitness.
        aWitness giveClueTo: self.
        self decreaseTimeBy: self penaltyTimeForAskingClues.
```

Ahora, sabiendo que cada testigo tiene como máximo dos pistas, el persuasivo responde la siguiente manera:

```
askTo: aWitness
        self validateWitness: aWitness.
        """ Le pide todas las pistas, es decir, le pide dos """
        aWitness giveAllCluesTo: self.
        self decreaseTimeBy: self penaltyTimeForAskingClues.
```

Depende del detective como pedirle pistas al testigo, y el testigo responderá de manera distinta dependiendo del mensaje que le llegue.

El detective puede viajar a otros países. Una vez que hizo todas las preguntas necesarias, puede viajar a un país limítrofe o no. ¿Como saben los países a que países tiene como limítrofe? En un principio, los países no tienen ningún país limítrofe: les tenemos que ir avisando a que países tienen en sus límites. Para eso, por ejemplo, podemos hacer lo siguiente:

```
uruguay addBorderingCountry: argentina.
```

Acá le estamos diciendo a Uruguay que Argentina es un país limítrofe. Ahora bien, esta relación es simétrica: Si Argentina tiene como país limítrofe a Uruguay, entonces Uruguay tiene como país limítrofe a Argentina. Para lograr que haya consistencia en esta relación, se manda este mismo mensaje a Argentina dentro del mismo método del mismo:

```
addBorderingCountry: aCountry
        (self isBorderingCountry: aCountry)
                ifFalse: [ borderingCountries add: aCountry.
                                aCountry addBorderingCountry: self ]
```

Los países limítrofes se van guardando en una *Set*. Ahora, podríamos haber hecho algo mucho más sencillo: hacer un mensaje que sea *#addBorderingCountry:* y que directamente agregue a los países adentro del set. Esto puede resultar ser una decisión de diseño peligrosa porque es posible que perdamos consistencia de la relación. Es preferible que los objetos sean los que se encarguen de cómo lo van a agregar y que la relación simétrica se mantenga siempre.

Para el detective, no es lo mismo viajar de Argentina a Francia que de Argentina a Uruguay. En el modelo, un viaje a un país no limítrofe cuesta más tiempo que viajar a uno que lo es. ¿Quién decide cuanto tiempo le sacamos al detective? El país al que vamos. Por ejemplo, si viajamos de Argentina a España:

```
aDetective travelTo: spain.
```

La forma en que le bajamos tiempo será de la siguiente forma:

```
decreaseTimeByTraveling: aCountry
        self decreaseTimeBy:
            self penaltyTimeForTravelling -
            (aCountry reductionTimeOfTravellingTo: currentCountry)
```

El mensaje *#reductionTimeOfTravellingTo:* se encarga de ver si tenemos que sacarle más tiempo en caso de que viaje a un país que no es limítrofe.

El detective puede seguir viajando, preguntando, hasta que se quede sin tiempo. Es más, puede ser que tenga tiempo y quiera culpar a cualquiera de los sospechosos. Para eso emite una orden de arresto al que cree que es el culpable. Para eso:

```
aDetective emitWarrant: aSuspect.
```

¿Qué sucede al emitir la orden? Como esto determina el juego (si es que se gana o se pierde), los sospechosos pueden devolver un "estado" de juego. Los sospechosos que no son culpables devuelven el estado de juego "Perdido" o "Lost" y los que son culpables devuelven el estado de juego "Ganado" o "Win". Si justo emitimos la orden al sospechoso correcto, el detective pasa de tener el estado de "En Progreso" a "Win" y en caso de que se haya equivocado de sospechoso pasa al estado "Lost".

El juego acá estaría terminado.

3 Conclusiones

- En un primer momento, a la hora de crear a los países había una especie de "paradoja": si quiero crear a los países con todos los testigos ya con pistas de otros países, esos países tendrían que haber sido creados con pistas de otros países... y así. Es por eso por lo que en el momento en el que creamos a una país, este mismo arranca con testigos que no tienen pistas (ni de sospechosos, ni de países). Por ejemplo:

  ```
  spain := Country    withName: 'spain'
                      andClues: #('jamon crudo' 'flamenco' 'sagrada familia').
  ```

  Si España termina siendo un país en el que estuvo el ladrón se le manda el siguiente mensaje:

  ```
  spain   giveWitnessesCountryCluesFrom: arrayOfCountriesWhereSuspectWas
          andSuspectCluesFrom: guiltySuspect.
  ```

  El país se encarga de "repartir" las pistas a los testigos. Si ese mensaje no se manda, el país se queda con testigos sin pistas. Es una manera de tener todos los objetos **completos** y sin colaboradores internos con *nil*.

- ¿Como conservamos los atributos de los sospechosos? Una idea podría haber sido guardar un array como *("blonde" "fit" "1.98" "brown")*. Si vos lees eso te preguntarás: ¿Qué tiene de rubio?, ¿Qué tiene de marrón? Para eso, podríamos mejorar la expresión de los strings para dejar más en claro que es, pero hay que tener mucho cuidado de no equivocarnos de expresión a la hora de programar. Para eso, a la hora de crear sospechosos, se utiliza el siguiente mensaje:

  ```
  someSuspect := Suspect  withEyeColor: 'green'
                          hairShape: 'long'
                          hairColor: 'blonde'
                          bodyType: 'fit'
                          gender: 'male'
                          height: '1.65'.
  ```

  Al crear el sospechoso, se guarda un diccionario que guarda todos estos atributos:

  ```
  initializeWithEyeColor: anEyeColor
  hairShape: aHairShape
  ```

```
hairColor: aHairColor
bodyType: aBodyType
gender: aGender
height: aHeight

attributes := Dictionary new.

attributes at: 'eye color' put: anEyeColor.
attributes at: 'hair shape' put: aHairShape.
attributes at: 'hair color' put: aHairColor.
attributes at: 'body type' put: aBodyType.
attributes at: 'gender' put: aGender.
attributes at: 'height' put: aHeight
```

A la hora de pedirle "pistas" al sospechoso, este utiliza las claves del diccionario y cada uno de sus valores para devolver de forma muy expresiva las pistas:

```
clues
        | clues |
        clues := OrderedCollection new.
        attributes keys do:
            [  :attribute | clues add: (attributes at: attribute ),' ',attribute ].
        ^ clues shuffled
```

Por ejemplo, al pedirle las pistas a *someSuspect*, se termina devolviendo:

```
an OrderedCollection(   'green eye color'
                        . 'male gender'
                        . 'fit body type'
                        . '1.65 height'
                        . 'long hair shape'
                        . 'blonde hair color')
```

Resultó ser una manera elegante y cómoda de procesar las pistas y atributos del sospechoso. Si se quieren agregan más características, es simplemente agregarle más claves al diccionario y, por supuesto, modificar el mensaje con que obtenemos a un sospechoso.

- *State* es una clase de poca relevancia en el modelo. Simplemente se la agrego para poder tener una manera de "reportar" el estado del jugador: si perdió, ganó, o el juego sigue en progreso. Podríamos haber representado el estado del jugador con un String como 'Win', 'Lose', o 'In Progress'. Resulta ser una opción que, a la hora de hacer los test, puede ser que perdamos un poco de control a la hora de hacerlos y resulta ser una manera elegante de representar el estado en el que el jugador se encuentra (el estado seria si perdió, gano o todavía está en 'investigación').