

The Sims - Aclaraciones y sugerencias

¿Quién hace qué? ¿Cómo se lo pido?

Algo importante a la hora de resolver el TP va a ser definir qué objetos son los encargados de cada responsabilidad, qué mensajes les enviamos, y con qué parámetros.

Por ejemplo:

“A los Sims Peleados con la vida les atraen aquellos otros Sims que estén tristes.”

Entonces yo podría definir:

```
>>leAtrae: unSim  
^ unSim felicidad<200.
```

Pero, acá hay algo que no cierra... Imagínense que un día se levantan, desayunan muy bien, el tren llega rápido, el bondi viene vacío, ustedes no pueden creer la buena suerte que están teniendo.

Luego, llegan a la facu y un amigo les pregunta “¿Qué te pasa? Estás triste”. Desconcertados por el comentario, alegan lo contrario y ante la insistencia su amigo justifica “ Vos estás triste porque no llegaste saltando y cantando”

Naturalmente lo que les surgirá pensar es “YO sé cuándo estoy triste y cuándo no, es algo interno”

Bueno, el pobre Sim se siente igual (Sí, recuerden que ellos también tienen sentimientos!). Quien sabe cuándo está o no triste es el propio Sim. Es su responsabilidad. Entonces, preguntémoselo a él.

```
>>leAtrae: unSim  
^unSim tieneMenosDe200DeFelicidad.
```

¡Ey! ¿No le íbamos a preguntar si estaba triste y ya? ¿No era que él sabía eso y bla bla...? Sí, es cierto, entonces el mensaje que le mandaremos será ese mismo:

```
>>leAtrae: unSim  
^unSim estaTriste: 200.
```

Y le pasamos la cantidad de felicidad que queremos porque... Ok, no, si lo pensamos bien vemos que si cada sim sabe cómo resolver esto, no es necesario pasarle ningún dato.

```
>>leAtrae: unSim  
^unSim estaTriste.
```

Y acá ganamos en varias cosas, de las cuales dos muy evidentes son:

- Mi código es más declarativo. Enuncio qué pretendo y no cómo se resuelve.
- Si el día de mañana los locos de PdeP se levantan cruzados y para la entrega 2 nos dicen “los Sims buenazos nunca están tristes porque la bondad alegra la vida”, el código que hay que tocar es mucho menos.

¿ Y cómo se relacionan tantos objetos? D:

También es importante definir la estructura de nuestros objetos. Por ejemplo, los Sims trabajan. Entonces, ¿los Sims tienen un trabajo? ¿cada vez que trabajan les digo dónde? ¿Los trabajos conocen a sus empleados?

Entonces acá lo que me conviene es preguntarme “¿Qué quiero poder hacer con mi sistema?” “¿Qué tiene más sentido?”

Entonces lo primero para cuestionarnos es: ¿La relación entre éstos es fuerte, estructural o débil? Es decir, ¿Es algo que queremos que se mantenga en el tiempo, que un sim tenga un trabajo relativamente fijo, o es algo que queremos poder cambiar cada vez que el Sim trabaja?

En principio, semánticamente tiene sentido que el sim tenga un trabajo más o menos estable. Y por otro lado, no hay ningún requerimiento que explicita lo contrario, entonces antes que tomarnos el trabajo de andar pasándole siempre el trabajo como parámetro, preferimos una relación fuerte. Finalmente vemos que para otro requerimiento necesitamos saber si algunos sims trabajan en el mismo lugar, por ende esta relación no puede ser tan débil, es estructural.

Ahora, es estructural, OK, todo muy lindo, pero... ¿quién conoce a quién? Lo primero que vemos es que necesitamos que el Sim haga cosas con su trabajo (trabajar :P) y no a la inversa (Por ejemplo, en ningún lado necesito conocer a los trabajadores de un trabajo). Entonces no tiene mucho sentido que un trabajo conozca a sus trabajadores, eso nos complicaría a la hora de que los sims trabajen.

A veces puede ser que dos objetos necesiten conocerse mutuamente. No está mal esto, pero sólo si es realmente necesario.

Recuerden que no todos los atributos de un objeto estarán listados al principio del enunciado.

Comportamiento variable

Hay muchos casos en los que el comportamiento de los sims varía dependiendo de diferentes características. Por ejemplo para las valoraciones antes de Paradigmas ustedes podrían haber hecho algo como:

```
>>valorar:unSim
^self esSuperficial iftrue:[ unSim popularidad *20] ifFalse:[ self esPeleadoConLaVida
ifTrue:[ 0] if False:...]]]]]
```

Y tan largo y feo que ni ganas de terminar de escribirlo me dan.

Hemos visto ya herramientas que nos ayudan a dirimir estas cuestiones sin echar mano de múltiples cláusulas de if's. Aprovechémoslas :D.

Desempleados

¡Oh, no! Los niveles de desocupación en SimCity están por las nubes y su alcalde enfrenta turbas iracundas frente al palacio de gobierno. Mientras tanto, un grupo de programadores se devana los sesos pensando cómo resolver qué pasa cuando un Sim está desempleado.

Una forma de resolverlo sería decir:

```
self trabajo ifNil: [ Me rasco :D]
```

Sin embargo si lo hacemos así estamos distanciándonos de pensar las cosas de forma objetosa. Cuando José Smalltalk armó su lenguaje, hizo un objeto nil que cosificara a la nada misma. Esa nada es un poco genérica, mucho a nosotros no nos sirve. Pero tal vez podríamos pensar en cosificar nuestra propia nada, adaptada a nuestro caso...

Ataques de Celos

Nunca superes a un Sim en algo, o éste jamás te lo perdonará... Siempre que un sim se pone celoso, toma determinadas medidas. Entre las medidas incitadas por los distintos tipos de celos hay algunas cosas que cambian, pero otras son comunes a todas. Recordemos el concepto de orden superior y tratemos de no repetir lógica innecesariamente.

Manejando colecciones

A veces tenemos objetos que tienen colecciones. Si les queremos pasar una colección nueva con varios elementos no está mal hacer:

```
pirulo amigos: { jose . juan . tuVieja}1
```

Pero si queremos agregarlos de a uno está bueno hacer un mensaje para ese fin y encapsular el manejo de la colección:

¹ Recordar que el tipo de colecciones definidas entre {} tiene tamaño fijo, no se les puede agregar elementos. Por lo que pueden ser apropiadas para inicializar colecciones que no varían, pero no en otros casos.

pirulo agregarConocimiento: "juan estudia abogacía"