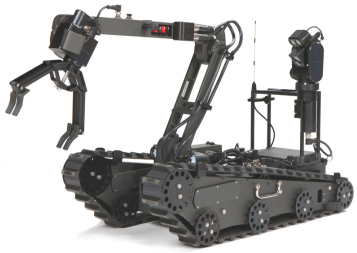
Trabajo Práctico 3
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 1er cuat. 2014

Fecha de entrega: 26 de junio



Introducción

Los departamentos de Geología y de Computación de una importante universidad latinoamericana se unieron para desarrollar un proyecto de exploración de tierras de bajo impacto ecológico y moderado costo, utilizando sistemas de robots autónomos.

En una etapa inicial, una sonda robot identifica zonas donde posiblemente existan metales preciosos, tan sólo realizando mediciones con ultrasonido desde la superficie. En base a la información recolectada por la sonda, un centro de control confecciona un mapa con la ubicación de los posibles puntos de interés que deberán ser explorados. Tras utilizar un sofisticado algoritmo para determinar el orden en que conviene visitar estos puntos, el centro de control guía de forma remota un robot minero hasta cada uno de los mismos.

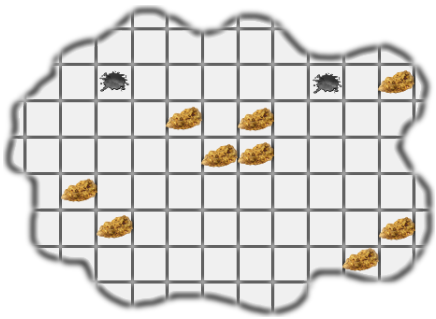


Figura 1: Mapa generado tras el análisis de la sonda

El robot minero se encarga de perforar la superficie del terreno para determinar qué es lo que hay efectivamente, y realizar otro tipo de mediciones según lo que se encuentre. Una vez procesadas las mediciones el robot envía un reporte al centro de control con los datos recolectados, siguiendo un formato estándar determinado por organismos internacionales para fomentar la interoperabilidad de los sistemas.

Material	**Orden de parámetros**	**Ejemplo**
Oro	Cantidad, Profundidad	`(13,256) --- ORO(100,1200)`
Petróleo	Densidad, Cantidad, Profundidad	`(20,-1800) --- PETROLEO(0.75,2000,3000)`
Ninguno	-	`(-200,500) --- NADA`

Figura 2: Formato de los reportes generados por el robot

Los mensajes comienzan con un par ordenado que indica la posición donde se realizó la medición, seguido de tres guiones y una palabra clave que identifica el tipo de material encontrado. En caso de que se hayan realizado mediciones extra para ese material, los resultados obtenidos se colocan entre paréntesis, en el orden que indica la figura 2.

Se desea implementar con programación orientada a objetos el posicionamiento del robot en el plano, el análisis del terreno y su interacción con el centro de control. Dado que el sistema podría desplegarse en terrenos no previstos, se desea que el diseño permita la fácil incorporación de nuevos tipos de elementos a registrar y que sea un buen modelo conceptual del problema para poder ser mantenido y extendido por otros equipos técnicos.

Ejercicios

Ejercicio 1

Modelar el plano donde se mueve el robot minero con la clase `Terreno`, que debe tener las siguientes características:

- Se deben poder crear instancias en base a un diccionario que indique qué hay en los puntos del plano no vacíos.
- No se deben poder crear terrenos con contenido inválido (es decir, con objetos que no representen algunos de los materiales conocidos hasta el momento).
- Un terreno debe informar qué hay en cualquier posición del plano, incluso si en esta no hay ningún contenido de interés.

Revisar la suite de tests `TestsTerreno`, donde se encuentran los nombres de estos mensajes y algunas pruebas del comportamiento esperado.

Ayuda: considerar el uso del mensaje `Object>>error:aString`, que interrumpe la ejecución con una excepción indicando la causa del problema.

Ejercicio 2

Extender el modelo para que el robot minero reciba órdenes con los movimientos a realizar para recorrer el plano. El robot no tiene conocimiento de la ruta que está desarrollando: sólo es capaz de realizar movimientos simples, que serán indicados por el centro de control tras el análisis del terreno.

Para lograr esto, hacer que el robot responda al mensaje `moverHacia:unaDireccion`, que incluye una de las cuatro posiciones en las que puede moverse (*arriba*, *abajo*, *izquierda* o *derecha*).

En la suite `TestsRobotMovimiento` se encuentran algunas pruebas del comportamiento esperado.

Ejercicio 3

Extender el modelo para que el robot pueda sensar el terreno en la posición actual. Para esto se debe implementar los mensajes:

- `reporte:objetoEncontrado`, que retorna el reporte a enviar al centro de control dependiendo de qué se haya encontrado en el terreno.
- `sensar`, que examina el contenido del terreno, registrando en la unidad de almacenamiento local lo que se encuentre y enviando el reporte correspondiente.

En la suite `TestsRobotSensado` se encuentran algunas pruebas del comportamiento esperado.

Ejercicio 4

Modelar con la clase `EjeDeCoordenadas` los ejes del plano en los que se mueve el robot. Estos objetos serán utilizados para manipular cada cordenada de su posición, de modo de acercarlo a un punto de interés. Para esto se debe implementar los mensajes:

- `obtenerPara:algoEnElPlano`, que averigua al posición sobre el eje de un objeto en el plano. Asumimos que los objetos en el plano responden a los mensajes `x` e `y`.
- `direccionCreciente` y `direccionDecreciente`, que representann las direcciones en las cuáles se mueve el robot según se incremente o disminuya el valor numérico de la coordenada correpondiente.
- `x` e `y`, que crean los dos ejes de coordenadas sobre los cuales se mueve el robot.

En la suite `TestsAjusteDeCoordenadas` se encuentran pruebas sobre el algoritmo que modifica una coordenada del robot tanto como sea necesario para que quede alineada con la de un punto de interés.

Ayuda: considerar el uso del mensaje `Object>>perform:aSymbol`, que envía mediante metaprogramación un mensaje sin argumentos a un objeto.

Ejercicio 5

Tras el análisis del terreno, el centro de control inicia una misión en la cual se le indica a un robot que visite por orden cada uno de los puntos de interés. Para modelar este comportamiento se debe implementar los mensajes:

1. `moverXHacia:unDestino` y `moverYHacia:unDestino`, que realizan los ajustes de coordenadas correspondientes para llevar al robot al punto de interés de turno.
2. `ejecutar`, que hace que el robot visite y sense cada uno de los puntos de interés.

En las suites `TestsMision` y `TestsCentroDeControl` se encuentran algunas pruebas integrales de cómo funcionan las misiones de exploración del terreno.

Pautas de entrega

El entregable debe contener:

- un archivo `.st` con todas las clases implementadas
- versión impresa (legible) del código, comentado adecuadamente
- **NO** hace falta entregar un informe sobre el trabajo

Se espera que el diseño presentado tenga en cuenta los siguientes factores:

- definición adecuada de clases y subclases, con responsabilidades bien distribuidas
- uso de polimorfismo para evitar exceso de condicionales
- intento de eliminar código repetido utilizando las abstracciones que correspondan

Consulten todo lo que sea necesario.

Consejos y sugerencias generales

- Lean al menos el primer capítulo de *Pharo by example*, en el cual se muestra a grandes rasgos cómo utilizar el SUTest.
- Exploren la imagen Pharo. Busquen inspiración en la clase `Boolean` y su estrategia de subtipado con las clases `True` y `False`.
- Pueden agregar tests propios (se recomienda agregar lo que utilicen para hacer sus propias pruebas).

Importación y exportación de paquetes

En Pharo se puede importar un paquete arrastrando el archivo del paquete hacia el intérprete y seleccionando la opción "`FileIn entire file`". Otra forma de hacerlo es desde el "`File Browser`" (botón derecho en el intérprete > `Tools` > `File Browser`, buscar el directorio, botón derecho en el nombre del archivo y elegir "`FileIn entire file`").

Para exportar un paquete, abrir el "`System Browser`", seleccionar el paquete deseado en el primer panel, hacer click con el botón derecho y elegir la opción "`FileOut`". El paquete exportado se guardará en el directorio `Contents/Resources` de la instalación de Pharo (o en donde esté la imágen actualmente en uso).