SISTEMAS INTELIGENTES
CURSO 2019/2020

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

PRÁCTICA 1: ALGORITMOS DE BÚSQUEDA CON ROBOCODE

OBJETIVO Y DESARROLLO DE LA PRÁCTICA

En esta práctica desarrollaremos un agente capaz de generar y seguir una ruta en un mapa definido mediante una malla con obstáculos. Para ello utilizaremos el software Robocode.

Robocode es un entorno de programación de juegos que permite definir batallas de tanques (robots). Existen distintos tipos de tanques, cada uno con un comportamiento dado. En cada batalla se introducen varios tanques, realizándose a continuación una simulación de su comportamiento hasta que la batalla termina. Se proporcionan entonces estadísticas del rendimiento de cada tanque.

Robocode permite definir nuevos tanques, así como su comportamiento, utilizando el lenguaje de programación Java. Aunque nuestro objetivo en esta práctica no es la definición de un tanque de combate, si que utilizaremos dicho entorno para definir y resolver un problema de búsqueda de caminos, así como simular la ejecución de los mismos.

La práctica se realizará preferentemente en grupos de tres alumnos, y se desarrollará en cinco sesiones de laboratorio. En la última de ellas, cada grupo deberá realizar una breve presentación de su trabajo (máximo 6 minutos). El proyecto desarrollado deberá entregarse adicionalmente a través de la tarea habilitada en el campus virtual.

El plan de trabajo tentativo contempla la consecución de los siguientes hitos al termino de cada sesión de laboratorio, aunque cada grupo puede seguir su propio ritmo de trabajo:

1. Instalación y toma de contacto con Robocode. Definición de un robot básico en Robocode.
2. Definición del problema de búsqueda mediante un generador aleatorio. Simulación del entorno.
3. Programación del comportamiento del robot mediante un algoritmo de búsqueda.
4. Simulación de la ejecución del camino generado. Depuración.
5. Presentación en clase del trabajo desarrollado.

Los materiales entregados deben corresponder al trabajo original de cada grupo. Cualquier otro material utilizado deberá ser debidamente acreditado.

HITO 1. INSTALACIÓN Y USO DE ROBOCODE

- El entorno Robocode puede descargarse de la siguiente dirección:

  http://robocode.sourceforge.net/

- Las instrucciones de instalación pueden encontrarse en:

  http://robowiki.net/wiki/Robocode_Download_And_Install

- Utilizaremos preferentemente el entorno de programación Eclipse. Las instrucciones para la creación de un proyecto pueden encontrarse en:

  http://robowiki.net/wiki/Robocode/Eclipse/Create_a_Project

- Puede crearse un nuevo robot creando la clase adecuada. Las instrucciones se encuentran en:

  http://robowiki.net/wiki/Robocode/Eclipse/Create_a_Robot

- Las siguientes instrucciones describen cómo puede acceder Robocode a los nuevos robots definidos:

  http://robowiki.net/wiki/Robocode/Eclipse/Create_a_Robot

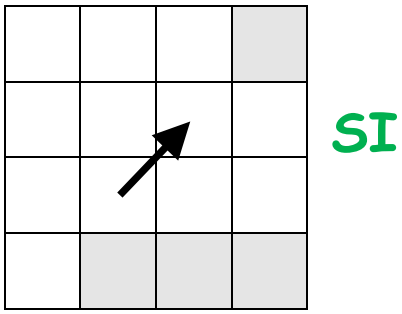
- Finalmente, podemos encontrar la API de Robocode en:

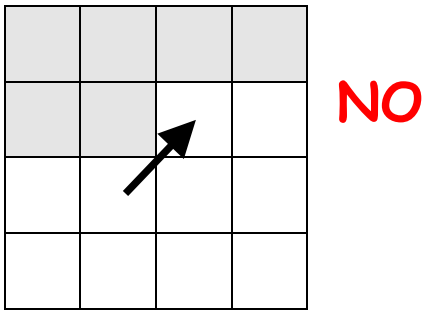
  http://robocode.sourceforge.net/docs/robocode/

HITO 2. DEFINICIÓN DEL PROBLEMA DE BÚSQUEDA

Definiremos un problema de búsqueda de caminos, habitual en muchos videojuegos, sobre una malla cuadrada con obstáculos. Consideraremos desplazamientos desde cada posición de la malla a los **8-vecinos** más cercanos (superior, inferior, derecha e izquierda, así como las posiciones adyacentes en las cuatro direcciones diagonales).

Los desplazamientos validos deberán corresponder a posiciones libres de obstáculos. Adicionalmente, **los movimientos diagonales sólo serán válidos si las posiciones a ambos lados de la diagonal están libres**. Dicho de otro modo, no se puede "recortar" la esquina definida por un obstáculo. Por ejemplo, el movimiento de la izquierda está permitido, pero no el de la derecha:





Consideraremos que el coste asociado a un desplazamiento en horizontal o diagonal es de 100 unidades, mientras que el de un desplazamiento en diagonal será de 142 unidades.

El tamaño de un tanque en Robocode es de 36 × 36 pixels. Utilizaremos por defecto un tamaño del campo de batalla de 800 × 600 pixels, que discretizaremos en celdas de 50 × 50 pixels. En cualquier caso, el tamaño del campo de batalla y de las celdas serán parámetros configurables en nuestro programa.

La generación del problema de búsqueda se realizará de la siguiente manera. Debemos desarrollar una **clase Problema** que genere una matriz (en el caso de las dimensiones anteriormente descritas sería de 16 × 12 celdas) con un número exacto de obstáculos dado (por ejemplo, 40 obstáculos) generados aleatoriamente. También se deberán generar aleatoriamente una posición inical y otra final válidas (libres de obstáculos) para nuestro robot.

La generación de problemas de búsqueda debe ser *reproducible*. Para ello utilizaremos un generador de números aleatorios de la clase Random. Proporcionaremos a nuestro programa los siguientes parámetros: la semilla del generador aleatorio, el número de filas de la malla, el número de columnas de la malla y el número de obstáculos. De este modo, siempre que se proporcionen los mismos parámetros, el generador deberá proporcionar *el mismo* problema de búsqueda (es decir, la misma malla de obstáculos y posiciones inicial y final).

Una vez definida la clase anterior, podemos simular el problema en el entorno Robocode completando la plantilla de código proporcionada en el campus virtual (**clase RouteFinder**). Esta clase debe definir una simulación Robocode de modo que:

- En la posición correspondiente al centro de cada celda con obstáculo se colocará un robot de la clase SittingDuck.
- En la posición correspondiente al centro de la celda inicial colocaremos nuestro propio robot (que en esta fase de la práctica podría limitar su comportamiento a girar sobre sí mismo).

HITO 3. PROGRAMACIÓN DEL COMPORTAMIENTO DEL ROBOT

En esta parte de la práctica definiremos un robot cuyo comportamiento consistirá en encontrar un camino óptimo para los problemas de búsqueda definidos anteriormente (**clase Robot**).

Una característica de Robocode es que no permite la comunicación explícita de información entre la aplicación Java y nuestro robot. Por ello, para asegurar que tanto el entorno como nuestro robot trabajan sobre el mismo problema, deberemos proporcionar a ambos el mismo número de obstáculos y semilla del generador aleatorio.

Una vez generado el problema de búsqueda, el robot deberá encontrar un camino óptimo entre las posiciones inicial y final de la malla de obstáculos. Debemos tener en cuenta que, al generarse aleatoriamente, no todos los problemas tendrán solución.

Para la resolución del problema se podrá utilizar:

1. Búsqueda primero en amplitud.
2. Búsqueda codiciosa (greedy search), es decir, usando como criterio de selección de ABIERTOS f(n) = h(n) y en caso de encontrar dos caminos al mismo nodo, conservando el antiguo. Se utilizará como heurístico h(n) **la distancia octil** hasta la posición final (movimiento en una malla 8-vecinos sin obstáculos)..
3. Búsqueda mediante el algoritmo A*, es decir, utilizando como criterio de selección de ABIERTOS f(n) = g(n) + h(n), y utilizando como heurístico la distancia octil hasta la posición final.

La valoración de la práctica será mayor a medida que se implementen correctamente, y de forma incremental, los tres apartados anteriores.

En cualquier caso los algoritmos deberán generar un árbol de búsqueda y devolver el camino solución, si este existe.

Representaremos el estado del robot mediante su posición en la malla. El árbol de búsqueda y la lista de nodos abiertos se implementarán obligatoriamente **según las estructuras de datos explicadas en clase.**

HITO 4. SIMULACIÓN DE LA RESOLUCIÓN DEL PROBLEMA

Finalmente, combinaremos las clases definidas anteriormente para simular la ejecución del camino. Tras encontrar el camino solución, el comportamiento del robot consistirá en ejecutar sus pasos uno a uno hasta llegar a la posición final.

Debemos tener en cuenta que, al igual que muchos programas gráficos, Robocode utiliza coordenadas cartesianas en su mapa de pixels. Si en la generación de nuestra malla de obstáculos hemos utilizado una matriz indexada por filas y columnas, deberemos tener en cuenta la posible transformación en la orientación a la hora de ejecutar las acciones correspondientes (giros y avances) en Robocode.

La depuración de un robot en Eclipse presenta ciertas particularidades, que pueden consultarse en:

http://robowiki.net/wiki/Robocode/Eclipse/Debugging_Robot

Una forma sencilla de saber qué está intentando hacer nuestro robot es escribir mensajes y consultar la consola asociada al mismo (accesible desde la simulación, pulsando en el panel que aparece a la derecha). Debemos tener en cuenta que los mensajes escritos antes de abrir la consola no aparecen en la misma.

https://stackoverflow.com/questions/30060859/cant-print-to-console-in-robocode

Como parte de la práctica, el robot deberá mostrar por consola las posiciones inicial y final del problema, el camino solución encontrado, **y el número de nodos seleccionados para expansión**.