UNIVERSIDADE DA CORUÑA

Grado en Ingeniería Informática
Sistemas inteligentes

Práctica 1: Estrategias de búsqueda
Curso 2021/2022

DOCUMENTACIÓN

El código del proyecto se suministra como proyecto de IntelliJ IDEA. En este documento se describe cómo utilizarlo. También se describe la estructura de clases, que deberá ser respetada en las implementaciones que se hagan.

Aclaración: No es necesario utilizar IntelliJ IDEA como IDE. Sin embargo, la entrega sí deberá presentarse como un proyecto de IntelliJ IDEA.

- **¿Cómo lanzar el código de ejemplo?**
  a) Si no dispones de IntelliJ IDEA, descárgalo de la página https://www.jetbrains.com/es-es/idea/download/. La versión Community es gratuita, de libre descarga y perfectamente válida para este proyecto.
  b) Abre IntelliJ IDEA e importa el proyecto. Para ello, lanza IntelliJ IDEA y en la pantalla inicial selecciona "Import". Te aparecerá una ventana en la que debes seleccionar el directorio en el que has descargado el código del proyecto. Una vez seleccionado, el proyecto se te abrirá en la pantalla principal de IntelliJ IDEA.
  c) Comprueba que puedes lanzar el código de ejemplo haciendo doble clic en el fichero "ejemplo/Main.java" desde la pantalla principal de IntelliJ IDEA. Se te mostrará el código de esa clase. Verás que hay un método "main", a la izquierda del cual se mostrará un icono ▶ verde. Púlsalo y se lanzará la ejecución de ese método. En la parte inferior de la pantalla se mostrará la salida por consola de la ejecución. Deberías ver un listado de pasos que se corresponden con la ejecución que aparece representada en las transparencias del TGR 2.
  d) Si necesitas más información sobre cómo utilizar IntelliJ IDEA, existen numerosos tutoriales en la página oficial: https://www.jetbrains.com/es-es/idea/resources/

- **Descripción del diseño**

  El proyecto que se suministra cuenta con las siguientes clases, que agruparemos atendiendo a su función, y que describimos a continuación:

  1. **Representación del problema:** El código nos ofrece tres clases que sirven de esqueleto para representar el problema (ver TGR 1). Para representar un problema dado, deberemos crear subclases de cada una de ellas (ver ejemplo).
     - **Clase Estado:** Define cómo se representa un estado para un problema dado. La clase estado es abstracta, por lo que no se pueden crear objetos de dicha clase directamente, sino que se deberá crear una subclase que herede de Estado y que implemente los métodos que requiere Estado. Será de esta subclase de la que podremos crear instancias (p.ej: ver clase EstadoAspiradora en `ejemplo/ProblemaAspiradora.java` y su uso en `ejemplo/Main.java`).
     - **Clase Acción:** Define una acción que se puede aplicar sobre un estado en un problema dado. La clase Acción también es abstracta, por lo que para crear objetos se deberán crear subclases que implementen los siguientes métodos:
       - ***esAplicable(Estado es):*** Método que indica si una acción se puede aplicar a un estado pasado como parámetro. Sirve para codificar las precondiciones de dicha acción.
       - ***aplicaA(Estado es):*** Este método devuelve el estado resultante de aplicar la acción sobre la que se llama al estado pasado como parámetro. Sirve para codificar el resultado de aplicar cada acción (modelo de transición).
       - ***getCoste():*** Devuelve el coste de la acción sobre la que se llama.
     - **Clase ProblemaBúsqueda:** Esta clase completa la definición del problema de búsqueda estableciendo el estado inicial y el test de meta. Es una clase abstracta, por lo que para crear instancias se deben declarar subclases específicas para cada problema (ver ejemplo). Las subclases que heredan de ProblemaBusqueda suelen declarar, a su vez, subclases de Estado y Accion. Consta de los siguientes métodos:
       - ***esMeta(Estado es):*** Implementa el test de meta. Indica si el estado pasado como parámetro puede considerarse una meta para el problema en cuestión. Es abstracta, por lo que debe darse una implementación específica para cada problema.
       - ***acciones(Estado es):*** Devuelve la lista de acciones aplicables al estado pasado como parámetro. Es abstracta, por lo que debe darse una implementación específica para cada problema.
       - ***resultado(Estado es, Accion a):*** Devuelve el estado sucesor resultante de aplicar la acción a sobre el estado es. Delega en la función *aplicaA* de la clase acción, por lo que no es necesario sobreescribirla.

  2. **Implementación de la estrategia de búsqueda:** En el código se suministran dos clases, una para estrategias de búsqueda no informada y otra para estrategias de búsqueda informada (que va acompañada de una clase para implementar heurísticas). Las clases son las siguientes:

- **EstrategiaBusqueda:** Es un interfaz que obliga a que todas las clases que lo implementan tengan un método para resolver problemas de búsqueda. Todas las estrategias de búsqueda no informada deben implementar este interfaz (ver `ejemplo/Estrategia4.java`). Métodos que contiene:
  - ***resuelve(ProblemaBusqueda p):*** Obtiene la solución (si es posible) al problema pasado como parámetro. Para ello sigue una estrategia de búsqueda sistemática que cada subclase debe implementar en este método.
- **EstrategiaBusquedaInformada:** Interfaz análogo a EstrategiaBusqueda pero que, al tratarse de búsquedas informadas, incluye el uso de una heurística (pasada como parámetro) en el método *resuelve*.
- **Heuristica:** Clase abstracta que dispone de un método para calcular el valor de la heurística de un estado pasado como parámetro. Las subclases que hereden de esta clase deberán implementar dicho método.

3. **Ejemplo:** Se incluye como ejemplo la implementación de la estrategia de búsqueda 4 (vista en el TGR2) para resolver el problema del mundo de la aspiradora (representado según se vio en el TGR1).
   - **ProblemaAspiradora:** Subclase de ProblemaBusqueda que contiene la formalización del problema del mundo de la aspiradora tal y como se vio en el TGR1. Define, a su vez, las clases EstadoAspiradora (subclase de Estado) y AccionAspiradora (subclase de Accion).
   - **Estrategia4:** Implementación de la estrategia básica de búsqueda vista en el TGR2. Es una implementación del interfaz EstrategiaBusqueda que implementa el método *resuelve* según lo visto en el TGR2.
   - **Main:** Clase principal que permite lanzar una ejecución. El método main instancia un ProblemaAspiradora especificando un EstadoAspiradora como estado inicial. Por otra parte, instancia un objeto Estrategia4, que es el que se utiliza para resolver el problema instanciado. Por último, se imprime por pantalla el resultado del método resuelve, que es el estado meta encontrado.