Memoria práctica 5

Alejandro Pascual Pozo y Víctor Yrazusta Ibarra

Apartado 1:

Para la implementación de las matrices dispersas nos hemos basado en la utilización de árboles rojo-negro. Estos están implementados en Java como TreeMap, y permiten que nuestra utilización de espacio sea mínima (O(n)) mientras que mantenemos O(ln(n)) para las operaciones esenciales (búsqueda, eliminación e inserción).

Elegimos esta implementación sobre una basada en HashMap principalmente por ahorrarnos las operaciones de rehash (que tienen como mínimo O(n) suponiendo una implementación eficiente de la función hashCode) y para no reservar memoria adicional. Esto se debe a que, como vamos a usar estas matrices para almacenar agentes que se mueven constantemente, los tamaños de estos mapas van a ser muy poco estables y se van a tener que reajustar numerosas veces.

Nuestra matriz permite un acceso eficiente a las filas, algo peor para las columnas.

Apartado 2:

Hemos aprovechado la función asList de la matriz, que nos permite obtener la matriz como una lista de elementos, para facilitar la implementación de las otras dos funciones. En concreto en la implementación del método equals hemos usado la función para facilitarnos la comparación de las matrices, y en asListSortedBy usamos asList para obtener la lista de elementos y simplemente la ordenamos según el comparador.

Apartado 3:

En este apartado hemos seguido el guion sin mayores complicaciones, implementando tanto el simulador como el programa para probarlo tal y como vienen descritos.

Apartado 4:

Para este apartado hemos creado la clase Behaviour, que encapsula la información de un comportamiento, con una condición necesaria para la realización de la acción y una acción que también determina si se pueden seguir encadenando acciones.

Nuestro agente extiende la funcionalidad del agente básico, lo que nos facilita el tratamiento con estos agentes, tanto en las celdas como en los simuladores.

En el simulador, para ejecutar el comportamiento de los agentes en orden aleatorio nos valemos de la función shuffle de Collections.

El simulador de este apartado extiende el del anterior, ya que comparten cierta funcionalidad, sobrescribiendo solo la función run para ejecutar los comportamientos de los agentes. Añadimos getNeighboursAt para obtener los vecinos de una casilla.

Apartado 5:

Para implementar los estados lo que hacemos es crear una versión original de estos al usar el constructor básico. Estos estados estarán siempre asociados al agente original, ya que este nunca será añadido directamente al simulador. Después, cada vez que se hace una copia del agente, se hace también una copia de estado actual de este. Cuando un agente cambia de estado, se recurre a la versión original de los estados para realizar una copia del nuevo estado necesitado.

Apartado 6 (opcional):

Hemos implementado las características adicionales de este apartado en Agent, ya que les vemos un uso muy similar al de los comportamientos y vemos coherente que pertenezcan a la misma clase.

Para implementar las propiedades hemos utilizado HashMap, utilizando los nombres como claves. Para comprobar el correcto funcionamiento de estos, que deben ser copiados cuando se realiza una copia del agente, hemos utilizado el test Apartado6A.

Para implementar las interacciones hemos utilizado un sistema muy similar al de los comportamientos, pero definiéndonos nuestras propias interfaces funcionales que acepten una entrada de dos argumentos. También nos aseguramos de que el agente no interactúe consigo mismo. Para comprobar el correcto funcionamiento de las interacciones hemos utilizado el test Apartado6B, que añade hormigas y comida a una matriz 3x3 y muestra como las hormigas se alimentan cada vez que encuentran comida, siempre y cuando esta no se haya agotado.

Diagrama de clases: