UNIVERSIDAD DE GRANADA

/ UGR / decsai

Departamento de Ciencias de la Computación e Inteligencia Artificial

Técnicas de los Sistemas Inteligentes

Curso Académico 2024-25

Práctica 1: Desarrollo de agentes basado en técnicas de búsqueda heurística dentro del entorno GVGAI

La Práctica 1 consiste en desarrollar cuatro agentes, basados en cuatro técnicas de búsqueda diferentes (incluyendo técnicas de búsqueda no informada y de búsqueda heurística, tanto offline como en tiempo real), dentro del entorno GVGAI[1], que guíe a un avatar a resolver un juego en distintos niveles. El juego escogido es *Labyrinth Dual*, disponible con el índice 59 en los tipos de juego "*singleplayer*" que se pueden encontrar en el fichero "all_games_sp.csv" de GVGAI.

1. Descripción general del juego

Labyrinth Dual es un videojuego sencillo cuyo objetivo consiste en alcanzar un portal de salida recorriendo un mundo laberíntico. Para ello, en primer lugar, el avatar debe evitar las trampas presentes en el mapa (tocarlas supondría la muerte inmediata), y también es posible que necesite capturar/recolectar una serie de capas que permiten superar ciertos muros. Más concretamente, hay tres tipos de muros: grises (totalmente infranqueables), azul-verdosos (que necesitan que el avatar porte la capa azul para poder traspasarlos), y marrones (que necesitan que el avatar porte la capa roja para poder traspasarlos). El avatar solo puede disponer de una única capa en cada instante: parte de un vestuario estándar (sin capa) y, a medida que va avanzando, puede ponerse las capas que encuentre (para ello, sencillamente basta con pasar por la casilla en donde esté la capa). El ponerse una capa implica perder la otra (que desaparece y no puede volver a ser reutilizada), y puede haber más de una capa de cada tipo en el mapa.

Acciones: se pueden ejecutar 4 acciones de Movimiento (IZQUIERDA, DERECHA, ARRIBA, ABAJO). También se puede realizar la acción NIL (acción nula) que, en realidad, consiste en no hacer nada y que el avatar, como consecuencia, no realice ningún movimiento.

En la siguiente figura se puede observar la configuración de un posible mapa del juego *Labyrinth Dual*.

[1] Se puede descargar desde https://github.com/GAIGResearch/GVGAI/archive/master.zip

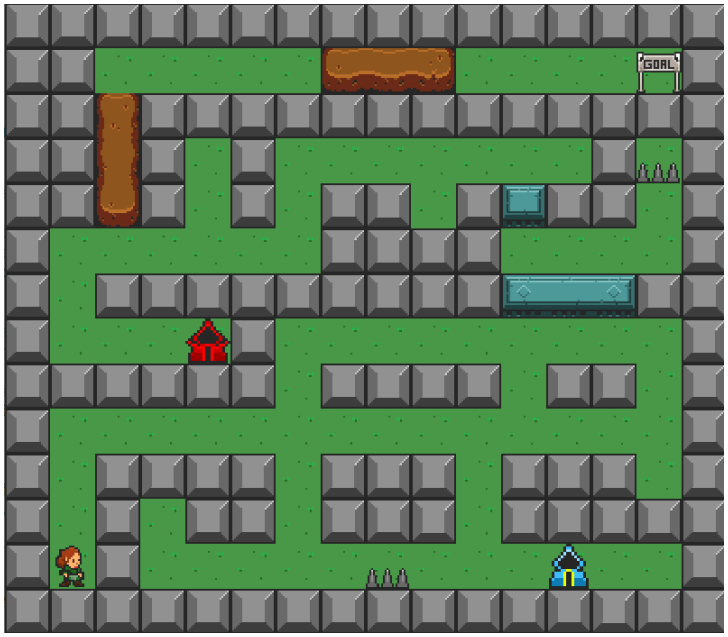

Figura 1: Mapa del primer nivel (levelIdx=0) de Labyrinth Dual (gameIdx = 59) en el framework GVGAI (General Video Game AI). Podemos ver el avatar en la esquina inferior izquierda, dos capas (una azul y otra roja), un portal de salida (en la esquina superior derecha del mapa), dos trampas (dos casillas más abajo del portal de salida, y cuatro casillas a la izquierda de la capa azul), y muros verde-azulados y marrones.

2. Descripción de la tarea a realizar

El objetivo de la práctica es que los estudiantes se familiaricen con el análisis e implementación de técnicas de búsqueda. Para ello, el alumnado deberá implementar los siguientes algoritmos de búsqueda:

- **Búsqueda no informada** (offline) con **Dijkstra**
- **Búsqueda heurística** (offline) con **A***
- **Búsqueda heurística** (online) **en tiempo real** con **RTA***
- **Búsqueda heurística** (online) **en tiempo real** con **LRTA***

Cada uno de estos algoritmos será ejecutado en mapas de diferente tamaño, desde mapas pequeños hasta mapas de dimensiones considerables y forma irregular (véanse Figuras 3, 4 y 5). El objetivo: verificar que las implementaciones realizadas son escalables y generalizan adecuadamente a diferentes configuraciones y tamaños.

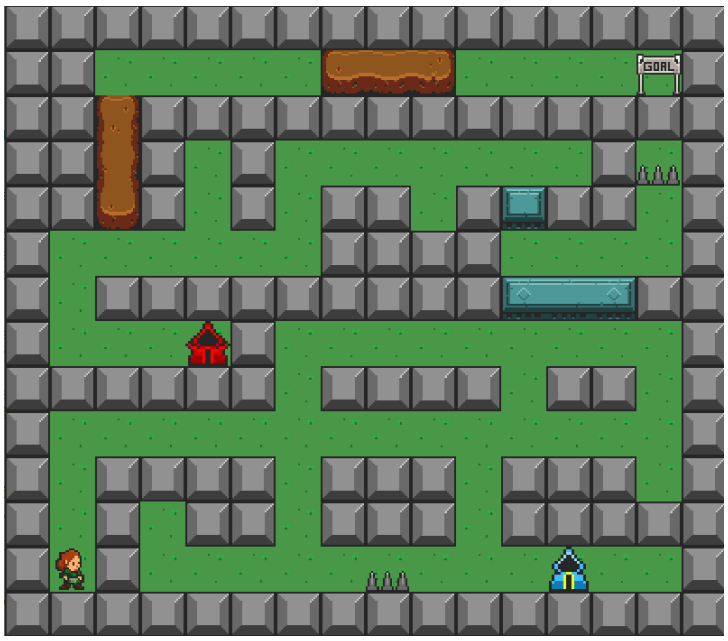

Figura 2: Ejemplo de mapa de tamaño pequeño (denominado labyrinthdual_lvl0.txt en los mapas proporcionados por defecto directamente en GVGAI)

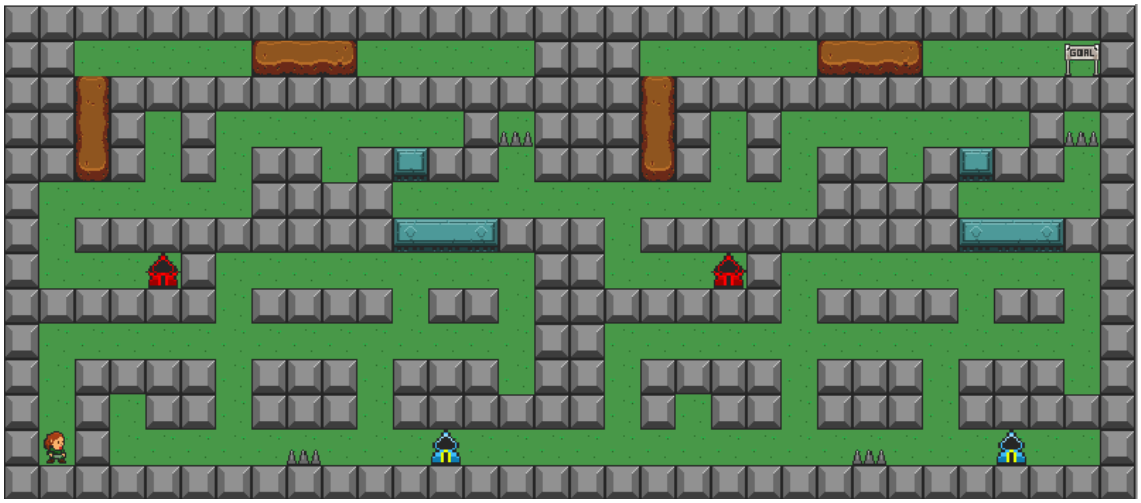

Figura 3: Ejemplo de mapa de tamaño mediano (denominado labyrinthdual_lvl5.txt en los materiales proporcionados a través de PRADO)

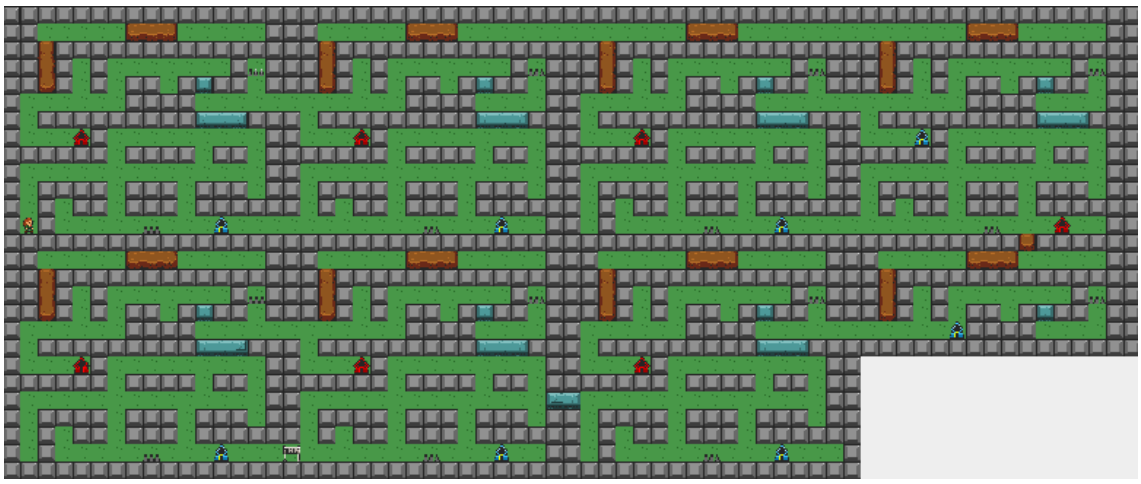

Figura 4: Ejemplo de mapa de tamaño grande (denominado labyrinthdual_lvl6.txt en los materiales proporcionados a través de PRADO)

Se recomienda encarecidamente que los estudiantes creen sus propios mapas para verificar el correcto funcionamiento de sus algoritmos. Debemos recordar que las soluciones propuestas por los estudiantes deben ser generalistas, es decir, dentro de la tarea en cuestión, deben permitir resolver correctamente otros mapas de dificultad aproximadamente similar (incluyendo, por ejemplo, mapas con una estructura menos laberíntica). De hecho, la evaluación será llevada a cabo, muy probablemente, en mapas distintos a los suministrados (aunque de similares características y dificultad).

3. Implementación

Toda la implementación debe desarrollarse dentro del siguiente **paquete Java** (que cada estudiante deberá crear al inicio de la práctica):

src/tracks/singlePlayer/evaluacion/src_APELLIDO1_APELLIDO2_NOMBRE

donde "nombre" y "apellidos" irán en **mayúscula** y sin guiones ni espacios (Java es case-sensitive). Dentro de este paquete, se definirán, al menos, las siguientes **clases Java** (case-sensitive):

- AgenteDijkstra
- AgenteAStar
- AgenteRTAStar
- AgenteLRTAStar

La implementación deberá respetar y/o considerar las siguientes restricciones, penalizando total o parcialmente la puntuación final obtenida en aquellos casos donde no se respeten o no se consideren:

- **Restricción de nomenclatura**. La nomenclatura anterior (nombre de paquete y clases) debe ser respetada **estrictamente**.

- **Restricciones en la ejecución.** Es importante tener en cuenta que cada ciclo de decisión está limitado de tal forma que la decisión de qué acción ejecutar en **cada *tick* debe tomarse en un tiempo no superior a 40ms**[2]. Además, **la ejecución se debe limitar a un máximo de 5.000 *ticks***. Por tanto, es importante que la implementación sea suficientemente eficiente, con una estructura de datos ligera y ágil en las operaciones requeridas, y que la definición del juego y de GVGAI permita la ejecución hasta 5.000 *ticks* (ver la presentación de la práctica para más detalles).

- **Restricciones en el constructor de los agentes**. Las operaciones realizadas en el constructor de los agentes se limitarán a acciones "menores" (de inicialización). En concreto, **toda acción relativa al método de búsqueda deberá realizarse dentro del método "act"**. Para ello, todos los agentes de búsqueda offline calcularán la ruta desde la posición actual del avatar hasta la casilla objetivo en una (o varias) ejecuciones del método "act", eligiendo como primera acción la primera casilla a la que desplazarse, y en las llamadas sucesivas a este método simplemente seguirán ejecutando los movimientos para desplazarse por la ruta ya calculada.

- **Restricciones de la heurística**. La función heurística usada por todos los métodos de búsqueda será siempre la **distancia Manhattan**.

- **Restricciones de expansión de vecinos**. Para la expansión de un nodo, el **orden de expansión** de sus **vecinos** será siempre el mismo: **Derecha, Izquierda, Arriba, Abajo**. Además, en caso de requerir un criterio adicional para ordenar los nodos expandidos, se usará un criterio histórico a modo de cola FIFO. Por ejemplo, A* ordena la lista de nodos abiertos según f(n) y, en caso de empate, según g(n). Si tras estas dos comparaciones persistiera un empate, se dará prioridad a los nodos más antiguos de la lista de abiertos. Nótese igualmente que si los vecinos de un nodo empatan en f(n) y g(n), se usará el orden anterior (derecha, izquierda, arriba, abajo) para introducirlos en abiertos.

- **Restricciones de mensajes**. Los agentes deberán **imprimir por pantalla únicamente** la información relativa a los algoritmos de búsqueda que se solicita en la Tabla 1.

[2] Si la acción seleccionada en el método *act* se decide entre 40-50ms GVGAI devuelve la acción nula; si el tiempo es mayor a 50ms se descalifica al agente.

4. Memoria

Una vez realizada la implementación de los distintos agentes deliberativos, los estudiantes deberán **ejecutarlos en los tres mapas indicados** (labyrinthdual_lvl0.txt, labyrinthdual_lvl5.txt y labyrinthdual_lvl6.txt) y completar la siguiente **tabla de resultados**:

Algoritmo	Mapa	Runtime (acumulado)	Tamaño de la ruta calculada	Nodos expandidos	Nodos Abiertos / Nodos Cerrados
Labyrinth Dual (id 59)					
Dijkstra	Pequeño				
	Mediano				
	Grande				
A*	Pequeño				/
	Mediano				/
	Grande				/
RTA*	Pequeño				
	Mediano				
	Grande				
LRTA*	Pequeño				
	Mediano				
	Grande				

Tabla 1: Tabla de resultados

Para rellenar la tabla anterior se deben tener en cuenta las siguientes consideraciones:

- La columna "*runtime*" hace referencia únicamente al tiempo total utilizado por el algoritmo de búsqueda para calcular el plan (sin contar el tiempo que cada agente invertirá en ejecutar los desplazamientos). Nótese que Dijkstra y A* únicamente realizan una llamada al algoritmo de búsqueda para calcular el plan hasta el objetivo, mientras que RTA* y LRTA* realizan múltiples llamadas. Es importante ser consciente de que si imprimimos resultados intermedios el tiempo de ejecución se ralentizará (y, dependiendo del tamaño de cada *print*, se puede llegar a ralentizar mucho la ejecución). Para medir el tiempo de cada una de estas llamadas, basta con usar estas líneas de código:

```
long tInicio = System.nanoTime();
// Código para el que se mide el tiempo
//  Ej: llamada A* que devuelve la ruta hasta objetivo
long tFin = System.nanoTime();
long tiempoTotalms = (tFin - tInicio)/1000000;
```

- La columna "tamaño de la ruta calculada" hace referencia al número de casillas transitadas por el agente durante la ejecución. Nótese que si un agente está en una casilla y, en el siguiente *tick*, decide quedarse en la misma, esa casilla "repetida" se incluirá en el tamaño de la ruta calculada. Es decir, lo que nos interesa es el número de acciones o *ticks*/*timesteps* necesarios para ir del origen al destino.

- La columna "número de nodos expandidos" hace referencia al número de nodos para los que se ha comprobado si son nodos objetivo durante la ejecución del algoritmo (nótese que, en algunos algoritmos como en RTA*, esta comprobación se puede realizar en múltiples ocasiones para el mismo nodo durante la ejecución del algoritmo).

- La columna "Nodos Abiertos / Nodos Cerrados" hace referencia al tamaño de las listas de abiertos y cerrados al finalizar la ejecución del algoritmo A*.

Nota: cada agente **solo debe imprimir por pantalla** la información que se pide para rellenar la Tabla 1, es decir, *runtime* (acumulado), tamaño de la ruta calculada y número de nodos expandidos (y número de nodos abiertos y cerrados en el algoritmo A*).

Finalmente, en la memoria se debe desarrollar la **respuesta justificada** de las siguientes preguntas:

1) Imaginemos que la representación del estado en nuestro juego (*Labyrinth Dual*) solo incluyese la posición de la casilla en que nos encontramos (coordenadas X e Y del *grid*). Por ejemplo, no se incluiría la capa como parte del estado. ¿Qué problemas y/o ventajas podría tener esta representación?

2) ¿RTA* y LRTA* son capaces de alcanzar el portal en todos los mapas? Si la respuesta es que no, ¿a qué se puede deber esto?

3) Imaginemos que debemos implementar búsqueda en profundidad como algoritmo de búsqueda para resolver este u otro problema. La forma más evidente de hacerlo es partir de una implementación de búsqueda en anchura y sustituir por una pila la cola que se emplee para almacenar los nodos abiertos. ¿Qué otra alternativa tendríamos a esta estrategia *naive*, y qué pros y contras tendría dicha alternativa?

4) El alumnado debe revisar el siguiente artículo científico: Hernández, C., & Meseguer, P. (2005). *LRTA*(k)*. In Proceedings of the 19th International Joint Conference on Artificial Intelligence (pp. 1238-1243). https://www.ijcai.org/Proceedings/05/Papers/0764.pdf ¿De qué modo LRTA*(k) se diferencia de LRTA*? ¿En qué consiste el algoritmo y cuáles son sus pros y contras en relación a algoritmos previos (como LRTA*)?

5) En el pseudocódigo del algoritmo A* existe el siguiente bloque:

```
if cerrados.contains(sucesor)
   and mejorCaminoA(sucesor): # menor g(n)
   • cerrados.remove(sucesor)
   • abiertos.add(sucesor) # actualizar g(n)
```

¿Bajo qué condiciones podemos estar seguros de que no vamos a entrar nunca dentro de este *if*? En otras palabras, ¿es siempre necesario implementar esa parte del algoritmo?

La respuesta a estas preguntas debe estar basada, allí donde corresponda y sea posible, tanto en las características teóricas de los algoritmos como en los resultados experimentales obtenidos. **Aquellas respuestas que no estén adecuadamente justificadas o desarrolladas podrán invalidar total o parcialmente aquellos ejercicios a los que afecten.**

5. Material a entregar

El material a entregar será un fichero ZIP con el siguiente contenido:

- La carpeta del paquete Java, que tendrá el mismo nombre que el definido anteriormente (src_APELLIDO1_APELLIDO2_NOMBRE). Dentro de esta carpeta estarán todas las clases Java correspondientes a los cuatro agentes deliberativos (y otras clases auxiliares usadas por éstos, en caso de ser necesarias), con las siguientes consideraciones:

  a) El **código** debe estar adecuadamente presentado y **comentado**, explicando los distintos aspectos de la solución propuesta por el estudiante de una forma clara: qué se hace, cómo se hace y por qué. **Un código insuficientemente comentado o con comentarios extremadamente deficientes podrá invalidar total o parcialmente el ejercicio al que afecte**.

  b) El código **no puede imprimir nada**, salvo lo especificado con anterioridad, correspondiente a la información para cubrir la Tabla 1.

- El fichero PDF de la memoria, con una **extensión máxima de 5 páginas**[3], donde se incluyan los **datos del estudiante**, la **tabla de resultados**, y las **respuestas razonadas** a las cuestiones planteadas.

[3] Esto representa un límite absoluto que incluye todo (desde una posible portada, a un posible índice, a una posible sección bibliográfica final, etc.).

6. Criterios de evaluación

La evaluación consistirá en la ejecución del **software entregado** para comprobar su efectividad y eficiencia en la resolución de distintos mapas. Durante esta ejecución, los profesores podrán usar **mapas diferentes a los proporcionados**, para comprobar que la implementación es correcta. La calificación de la práctica se realizará teniendo en cuenta las siguientes puntuaciones:

- Resultados de Dijkstra: 1.5 puntos
- Resultados de A*: 2 puntos
- Resultados de RTA*: 2 puntos
- Resultados de LRTA*: 2 puntos
- Memoria:
  - Pregunta 1: 0.5 puntos
  - Pregunta 2: 0.5 puntos
  - Pregunta 3: 0.5 puntos
  - Pregunta 4: 0.5 puntos
  - Pregunta 5: 0.5 puntos

Consideraciones sobre las calificaciones:

- La calificación sobre las distintas técnicas de búsqueda (Dijkstra, A*, RTA* y LRTA*) se basarán en los resultados reportados en la memoria, y los que los agentes obtengan en los mapas de evaluación empleados por los profesores (en caso de utilizar otros mapas distintos de los proporcionados como ejemplo con la práctica), así como en la documentación del código entregado.
- Una pobre presentación de la memoria podrá afectar negativamente a las puntuaciones obtenidas en las preguntas planteadas.
- MUY IMPORTANTE: En sí mismo, no supone ningún problema utilizar ChatGPT u otras herramientas de IA generativa como apoyo para el desarrollo de las prácticas. Se trata, sin duda, de herramientas de gran utilidad en la realización de múltiples tareas, y que nos permiten aumentar nuestra productividad en gran medida. No obstante, **cuando se empleen, se debe indicar con claridad y especificar de qué modo y en qué partes se han usado (y por qué). Lo contrario, puede suponer una penalización severa en la calificación recibida (incluyendo el suspenso de la práctica y/o de la asignatura).** Véase https://ceprud.ugr.es/formacion-tic/inteligencia-artificial (especialmente, la sección titulada "Recomendaciones e ideas para estudiantes").