uc3m

Enunciado de la práctica

BÚSQUEDA EN EL JUEGO DEL PAC-MAN

Curso Inteligencia Artificial
Grado en Ingeniería Informática
Curso 2019-20

1. Introducción

La tarea a realizar en esta práctica está relacionada con la parte de la asignatura dedicada a algoritmos de Búsqueda. **Esta práctica hay que realizarla de forma individual**. Utilizaremos una versión modificada del software proporcionado por la universidad de Berkeley (`http://ai.berkeley.edu/search.html`). El software concreto para realizar nuestra práctica se encuentra en Aula Global. El lenguaje de programación es Python (versión 2.7), aunque no es necesario conocerlo de antemano. El portal de Berkeley hay un tutorial sencillo sobre Python (`http://ai.berkeley.edu/tutorial.html`) y en la web hay cantidad de recursos relacionados con este lenguaje. El alumno no tiene que desarrollar código. La práctica está orientada a la comprensión y experimentación.
Es muy importante que la práctica se realice de forma individual. Es muy difícil que dos prácticas realizadas por distintos alumnos coincidan. En la corrección se utilizarán herramientas de detección de copia para verificarlo. Por otro lado, si un alumno no hace la práctica no aprende, que es el objetivo final de realizarla. Los profesores no vamos a ser flexibles respecto a ésto. Tolerancia cero frente a las copias para todos los alumnos implicados. Queremos alumnos honestos.

2. Descarga y ejecución del código

Después de descargar el código de Aula Global y descomprimirlo, se puede jugar al Pacman ejecutando el comando:

```
python pacman.py
```

Los ficheros del código relevantes para el alumno son:

- `search.py`: implementación de los algoritmos de búsqueda.
- `searchAgents.py`: implementación de los agentes basados en búsqueda.

Otros ficheros de código interesantes son:

- `pacman.py`: programa principal. En este fichero se describe el tipo *GameState*.

- `game.py`: lógica de las reglas de juego.
- `util.py`: estructuras de datos útiles para la implementación de los algoritmos de búsqueda.

El agente más sencillo incluido en `searchAgents.py` se llama `GoWestAgent`. Este agente siempre se dirige hacia el oeste:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

Y, claro, funciona mal cuando es necesario girar:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

En esta práctica veremos cómo construir agentes que se muevan de forma más inteligente en el escenario del Pacman.
El juego ofrece distintas opciones. La lista de opciones y sus valores por defecto se pueden ver haciendo:

```
python pacman.py -h
```

Todos los comandos de este enunciado aparecen en el fichero *commands.txt*.

3. Comprensión de algoritmos de búsqueda

En `searchAgents.py` se puede encontrar el agente `SearchAgent` que genera planes en el mundo del Pacman y después los ejecuta paso a paso. Para verificar que funciona correctamente ejecuta:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

Este comando le dice al `SearchAgent` que utilize el algoritmo de búsqueda `tinyMazeSearch`, implementado en `search.py`. Como se puede observar en su implementación, es un algoritmo programado *ad-hoc*. En `search.py` están implementados también los algoritmos de búsqueda: profundidad o *depth first search* (dfs), amplitud o *breadth first search* (bfs), Dikjstra o *uniform cost search* (ucs) y A$*$ (astar).
Los algoritmos de búsqueda se pueden ejecutar en distintos escenarios del Pacman. Por defecto, se realiza búsqueda en profundidad. Los siguientes comandos ejecutan un agente cuyo objetivo es comer una bola de comida utilizando búsqueda en profundidad:

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

Los distintos algoritmos se seleccionan utilizando la opción `-a`:

- Búsqueda en profundidad:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
  ```
- Búsqueda en amplitud:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
  ```
- Dijkstra:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
  ```
- A$*$ con distancia euclídea como heurística:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=euclideanHeuristic
  ```

En el interfaz se muestran en color rojo los estados explorados (un rojo más brillante indica que el estado se exploró antes). Para ejecutar más rápido se puede utilizar la opción `--frameTime 0`.
En esta parte se pide:

1. Estudia y comprende cómo están implementados los algoritmos de búsqueda en `search.py`.

2. Explica en la memoria de forma analítica la implementación de estos algoritmos. **Se debe explicar en qué consiste cada algoritmo. La explicación no debe hacer referencia a código concreto, aunque se puede utilizar pseudocódigo**.

4. Problema de encontrar la bola

En esta parte nos centraremos en el problema de que el Pacman encuentre una única bola situada en cualquier posición del escenario. Este problema es el que se resuelve en las ejecuciones anteriores. La definición de este problema se encuentra en el fichero `searchAgents.py`, concretamente en la clase `PositionSearchProblem`.
En esta parte se pide:

1. Estudia y comprende cómo está implementado el problema a resolver y cómo se relaciona la implementación en `PositionSearchProblem` con la implementación de los algoritmos de búsqueda en `search.py`.

2. Explica en la memoria de forma analítica cómo es este problema, de forma similar a cómo hemos hecho en otros ejercicios de búsqueda:

   - Cuál es el espacio de estados.
   - Cuál es el estado inicial y el test de meta.
   - Cuáles son los operadores, con sus condiciones de aplicabilidad y su resultado.

   (Puede ser útil poner trazas en el código python para imprimir variables en la salida estándar. Esto se puede hacer fácilmente con el comando `print <variable>`).

3. Realizar una comparativa de algoritmos de búsqueda y un análisis de su comportamiento, siguiendo las pautas que se indican a continuación. Se debe incluir en la memoria el resultado de realizar este apartado.

Para la comparativa de algoritmos, consideraremos búsqueda en profundidad, amplitud y A* con dos heurísticas: distancia Euclídea y distancia de Manhattan. En este apartado procede de la siguiente forma:

- Diseña 4 escenarios de complejidad incremental (más tamaño, distintas posiciones de las paredes) que te resulten interesantes para evaluar los algoritmos de búsqueda. Antes de realizar este apartado mira los ficheros del directorio `layouts` para comprender cómo se hace la definición de escenarios. Para ejecutar con un escenario concreto se utiliza la opción `-l`. Los escenarios deben ser diferentes de los ejemplos proporcionados y se deben explicar en la memoria.

- Ejecuta los 4 algoritmos en cada uno de los 4 escenarios e incluye en la memoria gráficas en la que se muestre una comparativa de los algoritmos para cada escenario y cada una de las siguientes variables: tiempo de ejecución, número de nodos expandidos y coste del camino encontrado. En total 12 gráficas. La Figura 1 muestra un ejemplo de comparativa de los estados expandidos para los 4 algoritmos en 3 escenarios proporcionados con el código.

- Analiza y explica los resultados obtenidos, considerando las propiedades teóricas de los algoritmos explicadas en clase.

- Intenta diseñar un escenario adicional en el que dfs encuentre la solución óptima y expanda menos nodos que A* con la heurística de la distancia de Manhattan. Incluye en la memoria la descripción del escenario y una explicación de por qué ocurre lo que se pide.

- Intenta diseñar un escenario adicional en el que A* con la heurística de la distancia de Manhattan expanda más nodos que bfs. Incluye en la memoria la descripción del escenario y una explicación de por qué ocurre lo que se pide.

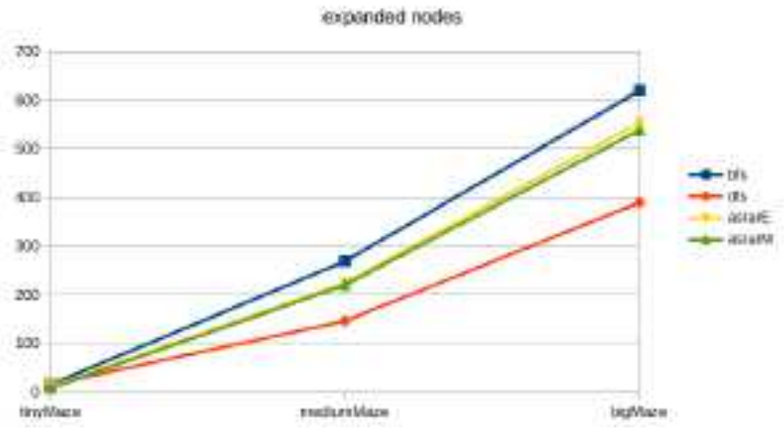

Figura 1: Ejemplo de gráfica.

5. Problema de comer todas las bolas

En esta parte nos centraremos en el problema de que el Pacman se coma todas las bolas situadas en el escenario. La definición de este problema se encuentra en el fichero `searchAgents.py` y corresponde con la clase `FoodSearchProblem`.

En esta parte se pide:

1. Estudia y comprende cómo está implementado el problema a resolver.
2. Explica en la memoria de forma analítica cómo es en este problema, concretamente:
   - Cuál es el espacio de estados.
   - Cuál es el estado inicial y el test de meta.
3. Para este problema se han desarrollado 3 agentes. El código de cada uno de ellos se encuentra claramente marcado en el fichero `searchAgents.py` (AGENT 1, AGENT 2 y AGENT 3). Estudia, comprende y explica en la memoria los algoritmos y heurísticas que utilizan cada uno de estos 3 agentes para resolver el problema.
4. Realiza una comparativa para comparar el comportamiento de los 3 agentes. Utiliza cómo mínimo los escenarios `trickySearch` y `bigSearch` (se pueden diseñar escenarios adicionales, incluso más sencillos para estudiar el comportamiento en este caso). Esta comparativa debe ser similar a la realizada anteriormente, comparando gráficamente los resultados en las variables tiempo de ejecución, número de nodos expandidos y coste del camino encontrado. A continuación se muestran los comandos para ejecutar cada uno de los agentes:

   AGENT 1:

   ```
   python pacman.py -l testSearch -p AStarFoodSearchAgent_ClosestFoodManhattanDistance
   ```

   AGENT 2:

   ```
   python pacman.py -l testSearch -p AStarFoodSearchAgent_ClosestFoodMazeDistance
   ```

   AGENT 3:

   ```
   python pacman.py -l testSearch -p ClosestDotSearchAgent
   ```

5. Analiza y explica los resultados obtenidos, considerando los algoritmos que utilizan cada uno de los agentes.

6. Entrega

La práctica podrá ser entregada a través del enlace que se publicará en Aula Global. La fecha de entrega es el **17 de mayo de 2020**. El nombre del fichero debe contener los 6 últimos dígitos del NIA del alumno (e.g., practicaIA-387633.zip). Este fichero debe contener **una memoria en pdf** (en Word utilizar la opción de exportar a pdf) que incluya:

1. Introducción.
2. Descripción de cada una las tareas realizas en la práctica, incluyendo en su caso descripción y explicación de las pruebas realizadas y sus resultados.
3. Conclusiones: conclusiones técnicas relacionadas con el desarrollo de esta práctica.
4. Comentarios personales: opinión acerca de la práctica, dificultades, etc.

Además de la memoria el fichero .zip al descomprimirse debe generar una carpeta con nombre *layouts-alumno* que contenga los escenarios diseñados.