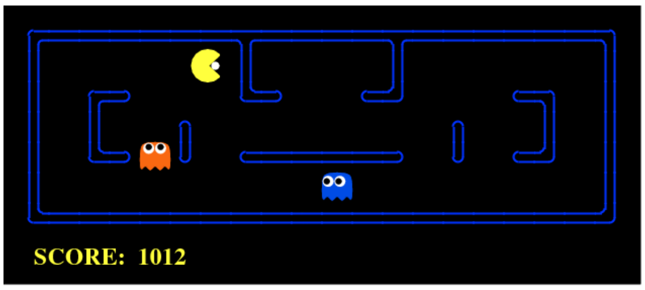
uc3m

Final Project Assignment

PATH SEARCH FOR THE PAC-MAN GAME

Artificial Intelligence
Computer Science Degree
Spring 2021



1 Introduction

In this project you will work with search algorithms applied for the PAC-MAN game. **This project must be done in pairs**. For purposes of this work, we make use of a modified version of the software provided by UC Berkeley (`http://ai.berkeley.edu/search.html`), which you can download from Aula Global. The programming language is **Python 3**. In order to make the code work, you need to install the libraries `numpy` and `future`.

You do not have to develop any code; the aim is to understand the code and experiment with it. **It is very important to do the project in pairs. Duplicate projects will be seriously penalized: both contributor and receiver will be punished. Automatic plagiarism detecting tools will be used to check plagiarism. Moreover, if you do not do the project, you do not learn, which is the aim of doing this project**. Plagiarism is unacceptable; professors will not be flexible on that. We want honest students.

2 Software installation

Make sure you have Python 3 is installed in your computer along with the libraries `numpy` and `future`. Once you have downloaded the code from Aula Global, you can play the PAC-MAN game by executing the following command: `python pacman.py`

The code files you need to know are:

- `search.py`: implementation of search algorithms.
- `searchAgents.py`: implementation of search-based agents.

Other interesting code files are:

- `pacman.py`: main file. This file describes a Pacman *GameState* type, which is use in this project.
- `game.py`: the logic behind how the Pacman world works.
- `util.py`: useful data structures for implementing search algorithms.

The easiest agent in `searchAgents.py` is the `GoWestAgent`, which always moves West.

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

Obviously, it does not work when turning is needed:

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

In this project, we will learn how intelligent agents can be implemented for the Pacman game.
The game has different options. You can check them by executing the following command:

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

You can find all the commands mentioned in this statement in *commands.txt*.

3 Understanding search algorithms

In `searchAgents.py` you can find the agent `SearchAgent`, which generates plans in the Pacman world and then executes them step by step. You can check this behavior by executing the following command:

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

This commands indicates the agent `SearchAgent` to use the search algorithm `tinyMazeSearch`, an *ad-hoc* algorithm implemented in `search.py`. In such file you can also find the following search algorithms: *depth-first search* (dfs), *breadth-first search* (bfs), Dikjstra or *uniform cost search* (ucs), and A* (astar).

These search algorithm can be executed in different Pacman mazes. If the command does not indicate any search algorithm, it will run a depth-first search. The following commands execute an agent whose aim is to collect a food dot using depth-first search:

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

To use a specific search algorithm, you can use the option `-a`:

- depth-first search:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
  ```
- breadth-first search:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
  ```
- Dijkstra:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
  ```
- A* with Euclidean distance as heuristic function:
  ```
  python pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=euclideanHeuristic
  ```

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). For a faster execution, you can use the option `--frameTime 0`.

In this section you are asked to:

1. Understand the code that implements the search algorithms in `search.py`.
2. Write an analytical explanation of such implementation. **You must explain what each algorithm consists of. Such explanation should not refer to the specific code, but you can use pseudocode.**

4 Problem 1: Finding a fixed food dot

In this section, we will focus on the problem where the Pacman has to collect a fixed food dot in the maze. The executions shown in the previous section solve this problem. You can find its definition in the class `PositionSearchProblem` defined in the `searchAgents.py` file.

In this section you are asked to:

1. Understand the code that implements the problem to solve, and figure out how the implementation in `PositionSearchProblem` connects with the implementation of search algorithms in `search.py`.

2. Write an analitycal explanation of such problem in a similar way as we have done the search exercises in class. That is:

   - Indicates which is the state space.
   - Indicates which are the initial state and the goal state.
   - Indicates which are the operator, their conditions of applicability and the result of their application.

   (It might be useful to inspect the state of a variable. You can do that by including the command `print(<variable_name>)` in the python code.)

3. Compare and analyze the behavior of the search algorithms following the guideline described next. For the comparison, consider the algorithms breadth-first search, depth-first search, and A* with Euclidean distance as heuristic function, and A* with Manhattan distance as heuristic function. Analyze and explain the obtained results.

Guideline for the comparison:

- Design 4 Pacman mazes with incremented complexity (higher size, different placement of walls) that you consider interesting to evaluate the search algorithms. To do that, check first the `layouts` folder. There you can find some example of Pacman mazes that might help you to understand how to do them. To execute a specific Pacman maze, use the option `-l`. The Pacman mazes to design must be different to the one provided in this project. Explain your designs.

- Execute each of the search algorithms in each of the Pacman maze designed. For each Pacman maze, generate a plot comparison among the search algorithms for each of the following variables: execution time, expanded nodes, and path cost. A total of 12 curves. Figure 1 shows an example of comparison between the expanded nodes for the four search algorithms within the Pacman mazes provided in this project.

- Analyze and explain the obtained results taking into account the theoretical properties of the algorithms explained in the classroom.

- Try to design an additional Pacman maze in which a depth-first search finds the optimal solution and expands less nodes than A* with Manhattan distance heuristic. Explain the Pacman maze designed and the reason why this happens.

- Try to design an additional Pacman maze in which A* with Manhattan distance heuristic expands more nodes than a breadth-first search. Explain the Pacman maze designed and the reason why this happens.

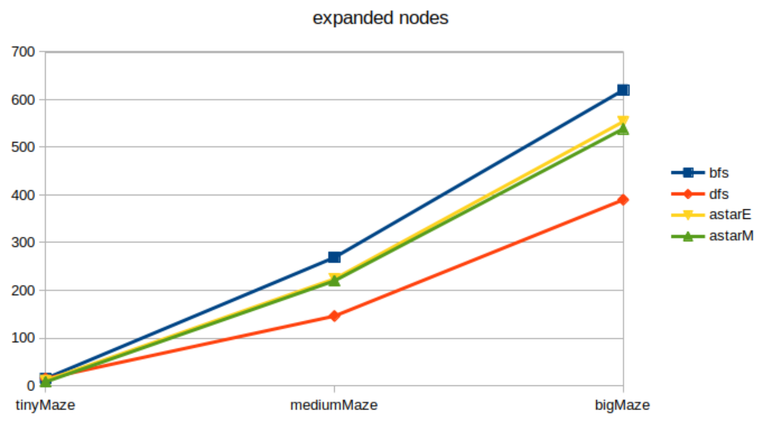


Figure 1: Example of plot diagram.

5 Problem 2: Eating all the food dots

In this section, we will focus on the problem where the Pacman has to collect all the food dots in the maze. You can find the definition for this problem in the class `FoodSearchProblem` defined in the `searchAgents.py` file.

In this section you are asked to:

1. Understand the code that implements the problem to solve.

2. Write an analytical explanation of such implementation. Concretely:

   - Indicates which is the state space.
   - Indicates which are the initial state and the goal state.

3. For this project, there are three different agents implemented. The code for each of them is clearly indicate in the `searchAgents.py` file (AGENT 1, AGENT 2 y AGENT 3). Understand and explain the algorithms and heuristics used for each of these agents to solve the problem.

4. Compare the behavior of the three agents. For the comparison, use at least the `trickySearch` and `bigSearch` boards. You could also design your own boards to perform this comparison. This comparison should be similar to the one performed in the previous section. Include a plot comparison of the results for the variables expanded nodes, execution times, and path cost. Next, you can find the commands to execute each of the agents:

   AGENT 1:

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

   AGENT 2:

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

   AGENT 3:

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

5. Analyze and explain the obtained results taking into account the algorithms used by each of the agents.

6 Submission

This project should be done **in pairs**. You must submit a zip file through a link that is posted in Aula Global. The zip file name must be *projectAI-student-code.zip*, where *sutdent-code* is the last six digits of the student's NIA (e.g., projectAI-123456.zip). The zip file will consists of:

- A report containing the following sections (**PDF** file):
  1. Introduction.
  2. Description and explanation of the tasks performed within the project: tests performed and obtained results.
  3. Conclusions: technical comments related to the development of this project.
  4. Personal comments: difficulties, challenges, benefits, etc.
- A folder, namely *layouts-student* containing the scenarios you have designed.

The deadline to submit the project is on May 19th at 11:55pm!