Project Report & Documentation

Introduction

Due to some *subjective misunderstandings* and *unwarranted conflicts*, no teams available for me and I had to work on a project **individually** and complete it within a **very short** period. I kindly request your understanding and approval for a solo project, without the necessity of a group involvement.

Course information

Course name	Project name	Teacher
Fundamentals of Artificial intelligence	Project 1 - Search	MSc. Quoc Huy Tran

Team members

Student ID	Full name	Tasks	Email
21120566	Huu Thuan Nguyen	All (*)	aswdqe1x@gmail.com

> (*) Any implementation included in this submission is done by myself, rejecting all forms of intellectual theft.

Folder structure

[documentation]

Containing all the contents of the project.

[input]

Divided into multiple subfolders, each representing a **level of the game**.

For each level, there are `.txt` files describing the **initial map and information of the game**.

[output]

Containing the **solution** respectively to the `[input]` folder.

A `.txt` file provides specific information regarding the solution approach, encompassing **Score**, **Number of steps** and **Detailed path**.

[source]

The **source code** of the project, written in **Python**:

- `Game.py`: This file contains various **Classes** that are used to represent and take over some logic and important functions of the game. Including `Monster`, `Food`, `Map`, `Snapshot` and `Solution`.

- `Helpers.py`: This file contains various **Constants** and **Functions** that are used to support the project and algorithms. Including `EMPTY`, `WALL`, `FOOD`, `MONSTER`, `PACMAN`, `PATH`, `DIRECTIONS`, `CORNERS`, `SURROUNDING` and `VISIBILITY`.
- `Search.py`: This file contains various **Search Algorithms** that are used to solve the game. Including `BFS` and `DFS` for different levels.
- `Visualizations.py`: This file contains various **Functions** that are used to visualize the game and the solution. Including `pygame` to simulate the solution and `matplotlib` to plot the comparison between methods.
- `main.py`: This is **the launcher**.

> Please refer to theses files for more information, they were commented thoroughly.

Usage

Install dependencies

```
cd source
pip install -r requirements.txt
```

Run the program

Arguments:

- `level:`: The level of the game, either `1`, `2`, `3` or `4`.
- `method`: The method to solve the game, either `bfs` or `dfs`.
- `visualization`: Whether run the visualization of the solution, either `True` or `False`.
- `comparison`: Whether run the comparison between methods, either `True` or `False`.

```
cd source
python main.py <level> <method> <visualization> <comparison>
```

Example

```
cd source
python main.py 2 bfs true true
```


pygame window
Step 17
Score: -16
Food remaining: 1

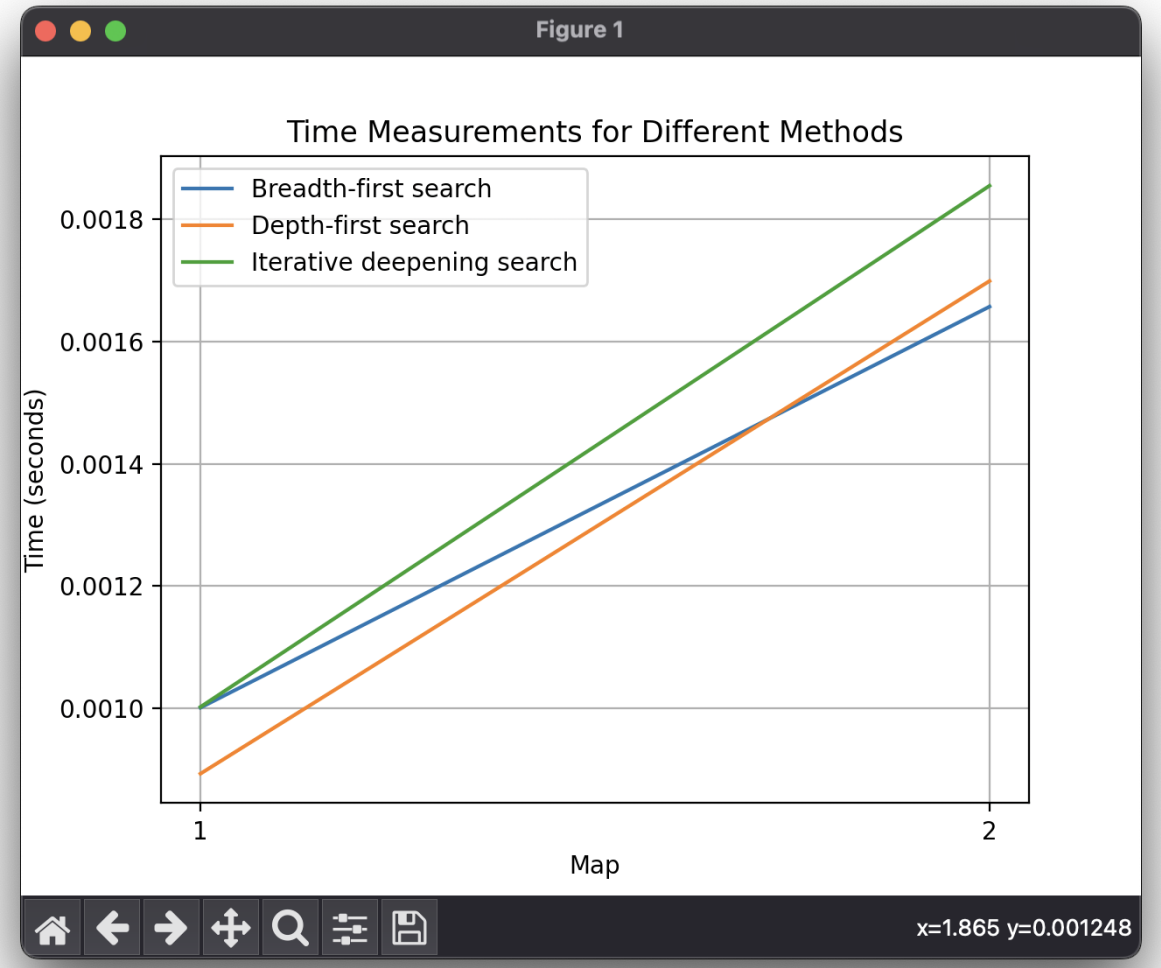
Figure 1
Time Measurements for Different Methods
Breadth-first search
Depth-first search
Iterative deepening search
0.0018
0.0016
0.0014
0.0012
0.0010
Time (seconds)
1
2
Map
x=1.865 y=0.001248

Ideas & Approaches

Basically, the search methods are relatively similar to each other. They were introduced in the lectures already, therefore I will focus on the main **ideas** and **approaches** that I used to solve the game.

Ideas

Helpers

Helpers are constants and functions that are used to support the project and algorithms.

Some of very important functions are `DIRECTIONS` and `SURROUNDING`, it returns a set of positions that are in the corresponding direction, surrounding and visibility of the given position.

> There are some definitions that I used in the project, apologies if it is not what you expected in the project description.
>
> **DIRECTIONS**: The 4 directions that `Pac-Man` and `Monster` can move, including **Up**, **Down**, **Left** and **Right**.
>
> **SURROUNDING**: The 8 positions that are surrounding the given position, including **Up**, **Down**, **Left**, **Right**, **Up-Left**, **Up-Right**, **Down-Left** and **Down-Right**. If `Pac-Man` is on one of these positions from a `Monster`, the game is over.

Game

`Game.py` contains 70% logic of the game.

Snapshot

One of the problem that I encountered is there are multiple `Food` and moving `Monster` on the map, it is little bit hard to visualize the solution step-by-step.

Therefore, I decided to divide the game state into multiple `Snapshot` objects, each representing a state of the game after a `Food` is eaten. Besides, it also stores the `path` that `Pac-Man` has gone, the list of `Monster` to track the positions that they have visited and the list of `Food` to track the eaten `Food`.

Solution

`Solution` object is simply an ordered list of `Snapshot` objects. With this approach, we can easily visualize the solution step-by-step.

Program flow

Step 1: Initialization

- Process the arguments.
- Read all maps from the `[input]` folder with the given level.

- Initialize time tracking variable.
- For each path, check if there is `Food`, then initialize the `Map` object.
- Do search at `Search.py`

Step 2: Search

- Initialize `Solution` object, `path` for the solution, `monsters` for tracking their positions, `eaten` for tracking the eaten `Food`, and `time_data` for tracking the time.
- While `map.food_remaining()`, we knows there are foods left on the map, then we will execute one of search methods until find a `Food`, it is the nearest `Food`.
- If the argument `comparison` is passed, then we will execute all other methods for `time_data` tracking.
- If there is no `path` found, **exit the program**.
- Create a `Snapshot` object to store the current state of the game, then modify the `Map` object to clean and correspond it for the next `Food`.
- Update `Food` tracker and `Map` for the next phrase.

Step 3: Visualization & Comparison

- If the argument `visualization` is passed, then we will display the solution step-by-step by visualizing each `Snapshot` individually.
- If the argument `comparison` is passed, then we will plot the comparison between methods, including **Time** and **Map**.

Important notes

In map `3` and `4`, method `DFS` and `IDS`, there are conflicts that `Pac-Man` and `Monsters` behave weirdly, I have tried to fix it in time but it is still not working properly. Therefore, I recommend to use `BFS` and to be careful when trying to **run** those methods or **compare** them.

Summary

No.	Specifications	Total Scores	Estimated Scores
1	Finish level 1 successfully	15%	15%
2	Finish level 2 successfully	15%	15%
3	Finish level 3 successfully	10%	5%
4	Finish level 4 successfully	10%	5%
5	Graphical demonstration of each step of the running process	10%	10%
6	Generate at least 5 maps with differences in the number and structure of walls, monsters, and food	10%	10%
7	Report the algorithm, and experiment with some reflections or comments	30%	30%

No.	Specifications	Total Scores	Estimated Scores
	Total	**100%**	**90%**