*Advanced Program in Computer Science*

*CS420 - Artificial Intelligence*

**Project Report**

Student 1: Phạm Thế Quyền - 1651029

Student 2: Nguyễn Minh Nhựt - 1651027

1. **Project Overview:**
2. ***Assignment Plan:***

**-** Phạm Thế Quyền: Visualization, Part of Level 3 and Level 4, Map Design, Report

- Nguyễn Minh Nhựt: Level 1, Level 2, Level 3 and Level 4, Report

- The project is accomplished within 3 weeks

1. ***Environment:***

**-**Python 3.7.2 and pygame Library, on any Operating System, as long as pygame is install

1. ***Level of Completion:*** all Levels are 100%
2. ***References***: #Quyền add Ref here:

- Slides for Pseudo-code of BFS and Hill-climbing

- pygame Documentation

1. **Algorithms**
2. ***Level 1 and Level 2:***

These two levels look different, but in fact they are all the same. In level 2, the monsters do not move, hence their role in the game is no different from barriers. Therefore, we apply the same algorithm to both of them

There are a variety of search algorithm applicable to these two level. However, we choose BFS, with some heuristics tweak over others:

1. Depth First Search (DFS) does not yield optimal path, and thus not chosen.

2. Iterative Deepening Searching (IDS) is seemingly the best candidate for these levels. However because level 1 and level2 are supposed to be easy, the map is small, IDS’s efficiency does not differ much from that of BFS. On the other hand, more difficult maps usually include more barriers. In such maps, using Manhattan distance heuristics to guide IDS can have negative effects - increase running time, because the inner DFS tends to hit barriers more often. This also means that IDS have to run more inner DFS’s, rendering it inefficient compared to BFS.

3. A\* is also a good candidate, however, BFS has the advantage of early termination, while A\* has to run over all the nodes adjacent to the food. That’s not to mention the higher space complexity of A\*

Because of these aforementioned reasons, we use BFS, with a minor tweak: of all successors, prioritize the one with smallest Manhattan distance to the food. This is just a minor tweak, yet it does help to reduce some redundant loops.

1. ***Level 3:***

Because Pacman is blind in this level, the only applicable algorithm is Hill-Climbing. In an attempt to maximize score, we use heuristics that cause Pacman to move through as many cell in the maps as possible:

1. The map is divided into four parts, like Cartesian Coordinate Plane, and Pacman moves counter-clockwise, that is if it is in the first quadrant, it would prioritize moving to the second quadrants and so on.

2. If Pacman comes across the food, it would remember position of that food. This can be implemented by using a stack, that is, the nearest food Pacman come across is on the top. For every move, pacman will try to move toward the top of that stack to get the food, using Manhattan distance as heuristics (Hill-climbing)

1. ***Level 4:***

Monsters can move in this level, making a significant difference. Minimax is the best candidate for this, however since there are many monsters, each of which is an agent, the implementation of Minimax is too complex.

We therefore propose a simpler algorithm:

1. At first, Pacman would use BFS to get the nearest food. Pacman uses Manhattan distances as Heuristics, just like in Level 1.

2. If he gets blocked by a monster, he would again use BFS to find another path. If no path is found, Pacman willingly ends the game.

As for the monsters:

1. Every monster will gradually move toward cells of lower Manhattan distance to Pacman.

2. At each steps, there can be 2 cells of the same Manhattan distance to Pacman, the one with better heuristics for Pacman is preferred. That is, if there are two equally good successors to choose, the monsters will choose the one that appears to block Pacman’s path to food.