PROJECT PACMAN REPORT

March 28, 2019

Student ID: 1651062
Student: Mai Le Bao Linh
VNU - Ho Chi Minh City University of Science
Faculty of Information Technology

Contents

INTRODUCTION TO THE PROJECT

- Objective: The objective of this project is to apply the searching algorithms into a toy problem (Pacman) with different conditions and distinguish the differences in performance and how algorithms work.

- Testing environment: 5 different Pacman map with 3 maps for level 1 and 2, 2 maps for level 3.

- Involved algorithms: A-star Algorithm for level 1 and 2, BFS for level 3.

- The environment to compile and run the program: Windows (exclusively), run with Visual Studio.

LEVEL 1 & 2: APPLYING A-STAR ALGORITHM IN SOLVING INFORMED SEARCH PROBLEM

- Problem description:
- Level 1: Pac-man know the foodâĂŹs position in map and monsters do not appear in map. There is only one food in the map.
- Level 2: monsters stand in the place ever (never move around). If Pac-man pass through the monster or vice versa, game is over. There is still one food in the map and Pac-man know its position.
- From the description above, the author decided to implement the A-star algorithm in order to solve those problems since the location of the food was given, therefore heuristic measurement can be applied to find the path to the food. Because of the restriction in how Pacman can move comparing to other problems' objects (Pacman can only move in 4 directions, while others may go in at most 8 directions), the heuristic can only be calculated in 4 directions: up, down, left, right. The author applied 2 ways to calculate the heuristic h: Manhattan Distance h-function or Euclidean Distance h-function, which provide different outcomes of the result. Figure 1 shows the functions used in the algorithm.

```
float hEulCalculator(int posX, int posY, int targetX, int targetY) {
    //return sqrt(power(posX - targetX) + power(posY - targetY)); //Euclidean heuristic
    return abs(posX - targetX) + abs(posY - targetY); // Manhattan distance heuristic
}
```

Figure 1: Heuristic Functions

- After the calculation was complete, the heuristic h of each position will be sorted and rearranged in order to choose the best option. Since each step costs only 1, therefore the result depended only in the heuristic h.

```
float oriGn[4];
        int pos[4];
        memset(pos, -1, sizeof(pos));
        for (int i = 0; i < 4; ++i) {
            oriGn[i] = gn[i]; //Duplicate the g(n) array
        }
        int n = sizeof(gn) / sizeof(gn[0]);
        sort(gn, gn + n); //Sort the g(n) array
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                if (oriGn[i] == gn[j] && oriGn[i] > 0) {
                    pos[j] = i;
                    break;
                }
            }
        } //Rearrange the sorted array to recognize which step should be considered first
```

Figure 2: Sorting and rearranging the result of the heuristic

- Then, the best move will replace the previous position of Pacman to be the new position. The new position would be updated and drawn on the new map.

-The whole procedure will be repeated until Pacman found out the food. The flaw of this approach was that if the food is unable to reach, the Pacman wouldn't stop. More constrains is needed for this approach to stop the Pacman if the way to the food is not found.

```
int flag = 0;
        for (int i = 0; i < 4; i++) {
            switch (pos[i]) {
            case 0:
                    newX = pPosition.getX();
                    newY = pPosition.getY() + 1;
                    flag = 1;
                    break;
            case 1:
                    newX = pPosition.getX();
                    newY = pPosition.getY() - 1;
                    flag = 1;
                    break;
            case 2:
                    newX = pPosition.getX() - 1;
                    newY = pPosition.getY();
                    flag = 1;
                    break;
            case 3:
                    newX = pPosition.getX() + 1;
                    newY = pPosition.getY();
                    flag = 1;
                    break;
            default:
                break;
            }
            if (flag == 1)
                break;
        }
        int oldX = pPosition.getX();
        int oldY = pPosition.getY();
        if (count >= 4) {
            memset(visited, false, sizeof(visited));
        } //If there is no way to go, the visited map will be resetted to find a new way.
        visited[newY][newX] = true;
        setPacman(newX, newY);
        pPosition.setParent(oldX, oldY);
        map->setAttr(pPosition.getPX(), pPosition.getPY(), 0);
        map->setAttr(pPosition.getX(), pPosition.getY(), 4);
        map->move(pPosition);
        Sleep(500);
```

Figure 3: Update and draw

LEVEL 3: APPLYING BFS ALGORITHM IN SOLVING UNINFORMED SEARCH PROBLEM

- Problem description:
- Level 3: Pac-man will not see the foods if they are outside Pacman's nearest one-step. It means that Pac-man just only scan all the adjacent him (4 tiles). There are many foods in the map.Monsters still stand one place.
- From the description above, the author decided to implement the BFS algorithm in order to solve those problems since the location of the foods was unknown. To implement the BFS, 4 moving directions of Pacman will be considered as nodes connected with Pacman, 4 directions of those moving directions are also considered as nodes connected with them, and so on. The "graph" will expand until it meets the first food. Then the way will be traced back and draw. The process will end when Pacman cannot find any food or the food is too far.
- Due to the complexity of the algorithm, it will not be shown in this report.