Final Project Report

Group 12
Members:
B09201005　李秉鴻
B09901105　謝博揚
B09901107　郭宇誠
Topic: **PAC-MAN**

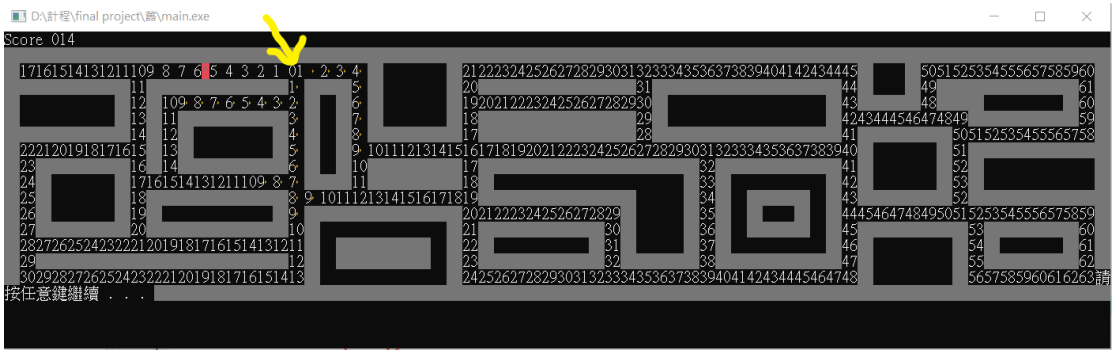
1. Introduction and Motivation

Inspired by the problem “Maze Exploration” on Online Judge, we decided to create a game that involves finding an algorithm which makes an object in the maze find another object (or exit). Therefore, we thought of the popular game “Pac-Man.” We drew our own maze and developed four modes: easy, medium, hard, and annoying. We also set up a leaderboard to store the score and time of top five.

2. Process and Difficulties

The very first problem we encountered is how to draw as we want on the window. We solved the problem by using the function `gotoxy` (after visiting a website).

Next, just like the problem “Maze Exploration,” we have to come up with an algorithm to enable the monsters to catch Pac-Man. The algorithm we use can refer to the following screenshot. The position that the yellow arrow points at is the Pac-Man’s position. We calculate the shortest distance between Pac-Man and every accessible block and record the distance with an integer `distance` (more details are in section 3). The monsters find Pac-Man according to the numbers. A monster observes the number of the neighboring blocks and choose to go to the block with the smallest number as its direction. This method effectively helps the monsters catch Pac-Man efficiently.



However, we encountered a problem. If every monster used the above algorithm to catch Pac-Man, the game would become boring because after a monster is 1~3 blocks behind Pac-Man, it follows Pac-Man continuously. That is, it never changes its path, and if Pac-Man doesn't stop or make a mistake, it will never catch Pac-Man successfully. Therefore, we add some conditions on the monsters' movements so that the monsters can approach Pac-Man from different routes or behave out of the player's expectation. The details are elaborated in the following section.

Finally, as the final presentation was around the corner, we wanted to beautify our interface by printing some big words on the window, but it would be staggeringly time-consuming if we had to specify where the colored blocks should be. Therefore, we wrote another program (setresult.cpp) to aid us. This program enables us to color a block by pressing 1 (or cancel by pressing 0) and move around by arrow keys, and print out the code after we finished drawing.

3. Features

Special Functions

```
int Maze::distance(int x_a, int y_a, int x_b, int y_b)
```

To make our monsters smarter, we implemented a function that returns the length of the shortest path between two points. It can help the monsters determine the direction which they move in.

The way we achieve it is by using a recursive algorithm. It starts at one point and then recursively goes leftward, rightward, upward, and downward in order until the length of the current path is not shorter than the minimal one that has already been found.

```
string Interface::verification(const vector< pair<string,pair<int,int> > >
&vec, int level)
```

In the class `Interface`, we have implemented a leaderboard. To achieve it, we needed to save game records to files. A natural idea is to prevent these files from being tampered with easily. So, we made the function `verification`. It generates a string according to the data so that we can verify whether the data has been tampered with.

Special Classes - Monster

One of the main parts of our program is the class `Monster` and its derived classes. In these classes, we have implemented many distinct monster moving algorithms. Now, we shall introduce them roughly.

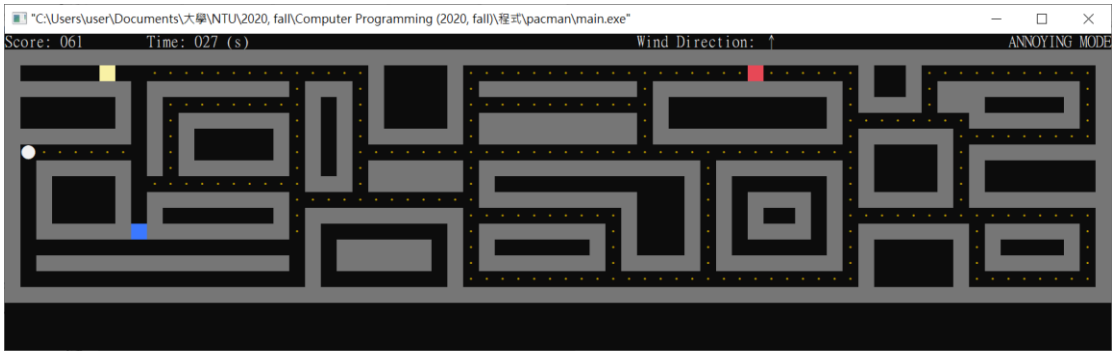
- Monsters in the base class `Monster` always move randomly.
- Monsters in `MonsterG` use the function `distance` mentioned above to determine the shortest path and move in the corresponding direction.
- Monsters in `MonsterL` use the function `distance` mentioned above to compare each distance between a monster and the Pac-Man. They usually choose the shortest path between them and the Pac-Man. However, there is an exception. When they think that on one path there is another monster, they tend to choose another path.
- Monsters in `MonsterH` sometimes move according to the relative position to the Pac-Man and sometimes move randomly so that the game will be more chaotic and more interesting.
- Monsters in `MonsterJ`, as those in `MonsterH` do, sometimes move according to the relative position and sometimes move randomly. The difference is that when they are far from the Pac-Man, they always move according to the relative position, instead of moving randomly.
- Monsters in `MonsterM` are used as an element of annoying mode, as we shall introduce them in part “Annoying Mode” below.

Annoying Mode

As one of the main features of our game, the annoying mode consists of several funny and special elements. Each time, one of them will happen randomly. Now, we shall introduce them.

1. Winds

When the wind is blowing, everyone will be pushed by the wind in the direction it blows.



2. Crazy Monsters

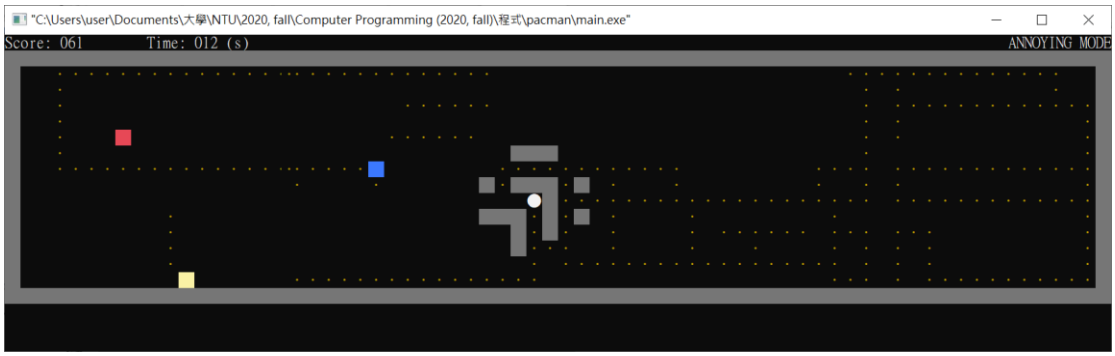
The monsters sometimes will become crazy and speed up or move across walls.

3. Unusual Arrow Keys

The arrow keys don’t correspond to the usual directions; instead, they correspond to the opposite directions to the directions they usually correspond to.

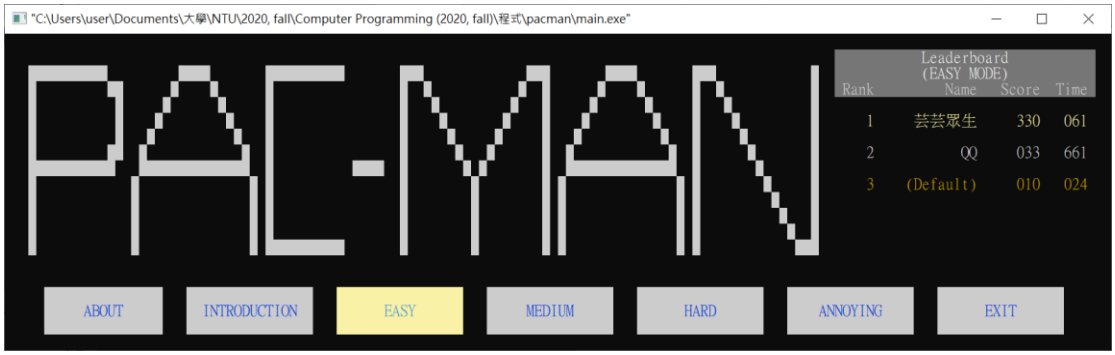
4. Invisible Walls

Only the walls that are sufficiently close to the Pac-Man can be seen.



Interface

Though we did not use SDL2, we still made endeavor to make our game look better. For example, as the graph below shows, there is a clear menu and a good-looking leaderboard on the homepage.



4. Reflections and Future Prospects

When doing this project, we realized the difference between coding alone and cooperating. For instance, when we wanted to add a new function, usually at least two team members had to modify their codes respectively. Also, our coding skills are not on the same level (QQ), so we sometimes had to wait for each other. Furthermore, choosing the topic is really a headache. On the first time of discussion, we spent 3~4 hours selecting the topic. We decided to give up the first two proposals because they are barely feasible and not flexible. We found that some teams changed their topics on the final presentation day, which made us feel that the time we spent on deciding the topic was absolutely worth it because we selected a goal that we can accomplish.

Finally, we felt that the process of doing the project is really interesting. The instant the basic version of the game successfully ran was the happiest (because we would not achieve nothing until the final presentation day XD). Some strange errors that occur during the initializing period always made us laugh. Most importantly, seeing that the codes written by us was really used in the game and successfully integrated together gave us a great sense of achievement. The project also enriched our experience and laid the foundation for bigger projects we may have to do in the foreseeable future.

As for future prospects, there are two main points. For one thing, the original Pac-Man game are not equipped with a fantastic interface. Also, in consideration of our time and skill, we decided not to utilize additional resources (such as SDL2 and OpenCV). In the future, we can make our game more exquisite by adding some animations and music. For another, we can add more features to annoying mode. For example, we discarded some ideas such as transporting points, bigger monsters, and the "kill screen" in the original game. Maybe we can simply write some new classes and enable multiple features in one gameplay in the future, enriching playing experience.

5. Work Distribution Chart

李秉鴻	謝博揚	郭宇誠
basic program	basic program	basic program
a powerful algorithm of a monster	algorithms of most monsters	
interface		interface
leaderboard		leaderboard
annoying mode		ideas of annoying mode
integrating all the codes	slides and presentation	

p.s. The basic program includes maze.cpp, pacman.cpp, and a part of main.cpp.