소프트웨어개발도구환경실습 최종프로젝트 보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

목차

1. 프로젝트 목표, 실험 환경에 대한 설명

이번 프로젝트에서는 maze project에서 학습한 길찾기 알고리즘을 기반으로 간단한 팩맨 게임을 만들어보기로 결정하였다. 팩맨은 1980년 5월 22일 일본의 회사 NAMCO에서 개발한 오락실용 게임으로, 주인공 캐릭터인 팩맨을 쫓아오는 유령을 피해 맵에 있는 모든 쿠키(=아이템)을 먹으면 승리하는 간단한 게임이다.

Windows 10, Visual Studio 2022, Debug/Release x64/x86 환경에서 정상적으로 실행되는 것을 확인하였고, 그 외의 환경에서는 테스트해보지 않았다. conio.h, windows.h 라이브러리 파일을 포함하기 때문에 Linux 운영체제와 표준 라이브러리로는 실행할 수 없다.

2. 각 변수에 대한 설명(=변수표)

전역 변수명	설명
int Map[25][25]	게임이 플레이되는 기본 맵
int vst[25][25]	recursive-dfs 미로 생성 알고리즘에 사용될 방문 맵
const int dy[], dx[]	상하좌우 이동 방향을 저장하는 상수 int 배열
int keyStroke	사용자의 키보드 입력을 저장하는 변수
int score	현재 점수를 저장하는 변수
int playing_round	지금 플레이 중인 게임의 라운드를 저장하는 변수
int gameOver, gameWin	게임이 승패 여부를 저장하는 변수
int SPECIAL_FLAG	스페셜 아이템을 얻었는지 여부를 저장하는 변수

random_device rd;
mt19937 gen(rd());
uniform_int_distribution<int> dis(0, 3);
	recursive-dfs 미로 생성 알고리즘에 사용될 난수, 난수를 사용하지 않을 경우 시드 값이 고정되어 항상 똑같은 미로가 출력됨

클래스명	설명
class Vertex	유령의 bfs 경로 탐색 알고리즘에 사용되는 정점 클래스, 좌표 값과 이동 방향(x, y, dir)을 저장한다.
class Path	유령의 bfs 경로 탐색 알고리즘에 사용되는 경로 클래스, 좌표 값과 이동 방향(x, y, dir), 현재까지의 이동 경로(vector<Vertex> path)를 저장한다.
class Ghost	유령을 저장하는 클래스, 유령의 현재 좌표와 이동 방향(x, y, dir), 현재 색깔과 최초 색깔(color, origin_color), 상태를 변경할지 말지를 나타내는 클록 신호(time), 현재 살아 있는지의 여부(isDead)를 저장한다.

3. 각 함수에 대한 설명(=함수표)

1) 유틸리티 함수

함수명	설명
void gotoXY (int x, int y)	<windows.h> 기반 함수, 입력한 좌표로 이동
void cursorHide()	<windows.h> 기반 함수, 화면이 깔끔하게 출력될 수 있도록 커서 깜빡임을 숨기는 함수
void textcolor (int foreground, int backfround)	<windows.h> 기반 함수, 글자의 색깔을 바꿈

2) 화면, UI, 초기화 함수

함수명	설명
void initPacMan()	게임 실행을 위한 다양한 초기 작업을 수행하는 함수
void dfs()	recursive-dfs 방식의 재귀적 미로 생성 함수
void printUI()	UI(라운드, 점수, 아이템 획득 정보)를 출력하는 함수
void printMap (int cx, int cy, Ghost ghost1, Ghost ghost2)	맵 정보(벽, 빈 공간, 아이템)를 출력하는 함수, 부자연스러운 화면 깜빡임을 줄이기 위해 캐릭터, 유령의 좌표를 입력받아 해당 좌표에서는 맵을 출력하지 않음
void isGameOver()	맵의 아이템을 모두 획득하여 게임이 종료됐는지 확인하는 함수
void win()	승리 화면을 출력하는 함수
void lose()	패배 화면을 출력하는 함수

3) 게임플레이 함수

함수명	설명
void getFood(int x, int y)	캐릭터의 좌표를 입력받아 음식을 먹고, 점수를 갱신하는 함수
bool isMovable (int x, int y, int dir)	개체의 좌표와 방향을 입력받고, 이동 여부를 확인하는 함수
void printEntity (int x, int y, const char* string, int color)	출력할 개체의 문자열과 색상, 좌표 값을 입력받아 출력하는 함수, 주로 팩맨 또는 유령을 출력하는 데에 사용
void moveEntity (int& x, int& y, int dir, const char* string, int color)	출력할 개체(유령, 팩맨)를 이동시켜 출력하는 함수, 개체의 이전 좌표를 비우고, 이동한 현재 좌표에 개체를 출력함
int direction()	dx[], dy[] 인덱스를 적절한 방향 키 상수(UP, DOWN, LEFT, RIGHT)로 바꾸는 함수
void meetGhost (int cx, int cy, Ghost& ghost)	유령과 팩맨의 충돌 여부를 판정하고 충돌 시 스페셜 아이템 획득 여부에 따라 적절하게 처리하는 함수

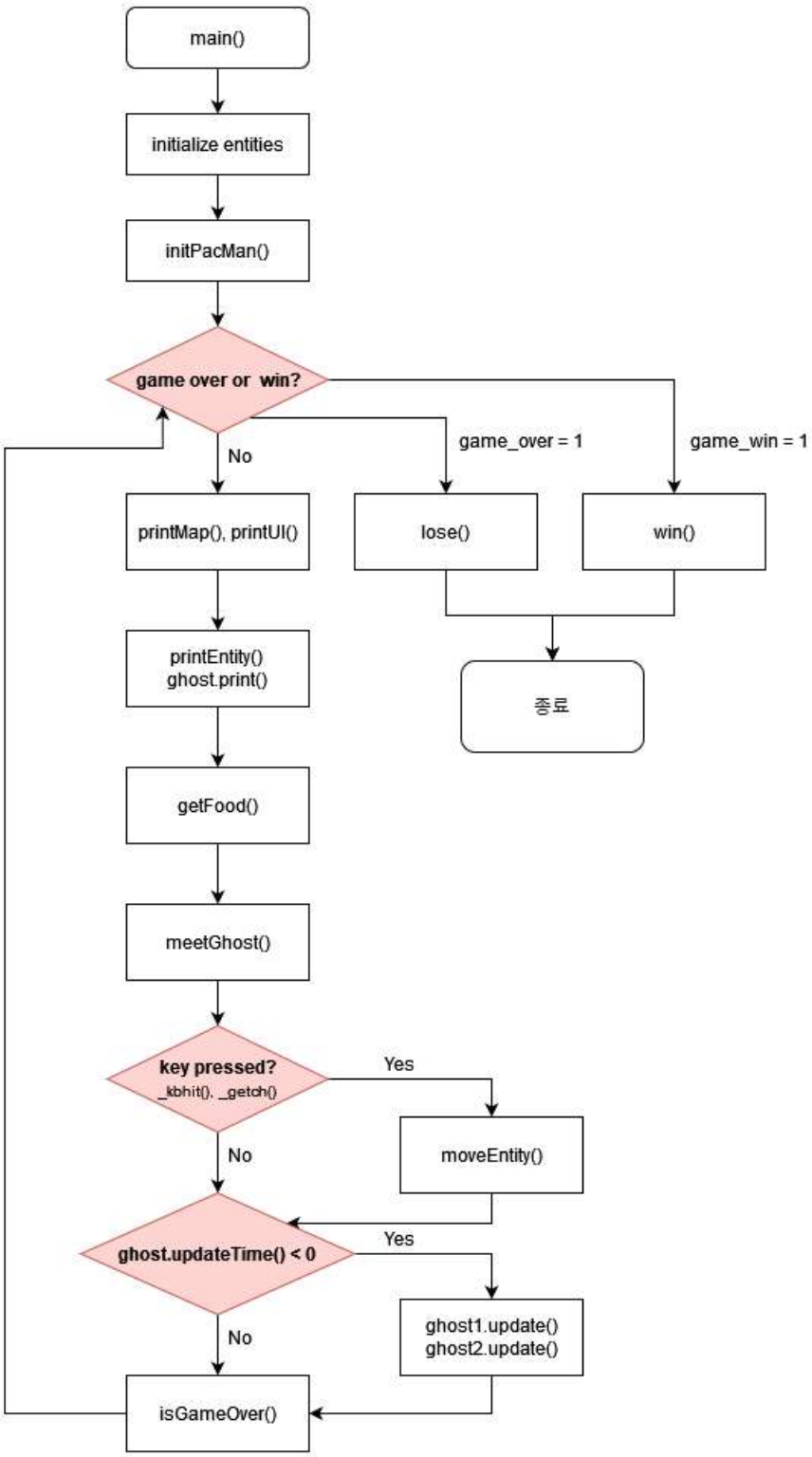
4) 각 클래스의 멤버 함수

Vertex class	설명
Vertex::Vertex (int pos_x, int pos_y, int pos_dir)	정점의 좌표를 입력받는 생성자 함수

Path class	설명
Path::Path (int pos_x, int pos_y, int pos_dir, vector<Vertex> prev_path)	정점의 좌표와 이전까지의 경로를 입력받는 생성자 함수
void path::copy int& pos_x, int& pos_y, vector<Vertex> &cur_path) const	입력받은 변수에 현재 Path 클래스 변수의 상태를 복사하는 함수, deep copy를 수행하기 위해 참조자로 입력

Ghost class	설명
Ghost::Ghost (int x, int y, int dir, int color)	유령의 현재 좌표와 방향 정보, 유령의 색을 입력받는 생성자 함수, 변하지 않을 변수인 origin_color 변수는 입력받은 색으로 초기화하며, time, isDead 변수는 0으로 초기화
int Ghost::getX(), int Ghost::getY(), int Ghost::getDead()	유령의 현재 x/y좌표, 죽었는지의 여부를 얻기 위한 함수
int Ghost::updateTime()	유령의 타이머를 갱신하고, 갱신된 타이머를 출력하는 함수
void Ghost::update (int character_x, int character_y)	유령의 타이머, SPECIAL_FLAG, 죽었는지의 여부에 따라 적절히 유령의 상태를 갱신하는 함수
void Ghost::move (int character_x, int character_y)	bfs 알고리즘을 이용해 입력받은 캐릭터의 위치에 대한 최단 경로를 찾고, 경로를 따라 이동하는 함수
void Ghost::die()	유령을 죽은 상태로 바꾸고, 유령의 좌표를 맵 끝으로 이동
void Ghost::print() const	printEntity() 함수로 유령을 현재 좌표에 출력

4. 플로우차트


main()
initialize entities
initPacMan()
game over or win?
game_over = 1
game_win = 1
No
printMap(), printUI()
lose()
win()
printEntity()
ghost.print()
종료
getFood()
meetGhost()
key pressed?
_kbhit(), _getch()
Yes
No
moveEntity()
ghost.updateTime() < 0
Yes
No
ghost1.update()
ghost2.update()
isGameOver()

main() 함수의 동작을 플로우차트로 그리면 다음과 같다. 먼저 게임 실행과 동시에 캐릭터 및 유령이 초기화되며, initPacMan() 함수에 의해 cursorHide() 등의 유틸리티 함수가 실행되고 무작위로 미로가 생성된다.

미로가 생성되고 나면 승리하거나 패배하지 않는 한 반복문이 계속해서 실행된다. 반복문이 한 번 수행될 때마다 맵과 UI, 개체의 위치와 상태 정보를 추적하여 출력하며, 만약 플레이어 캐릭터(=팩맨)의 좌표에 아이템이 있거나, 유령이 있을 경우 그에 맞춰 적절한 처리를 한다(=getFood(), meetGhost()). 만약 스페셜 아이템을 사용하지 않았을 때 유령과 마주쳤다면 game_over 플래그를 set한다.

플레이어가 값을 입력했다면 입력한 방향키에 맞게 개체를 이동하고, 유령의 update timer가 정해진 값보다 작아졌다면 유령을 이동시킨다. isGameOver() 함수로 맵을 탐색한 후 승리 조건을 만족했다면 game_win 플래그를 set한다. 게임이 종료된 뒤 flag에 따라서 win(), lose() 함수를 통해 승리/패배 화면을 출력한다. 만약 두 플래그가 모두 set되어 있을 경우 승리 화면을 출력한다.

5. 각 함수의 구현, 자료구조·알고리즘, 시간/공간 복잡도

함수들의 구현, 사용한 자료구조와 알고리즘, 시간·공간 복잡도는 다음과 같다. 단, 유틸리티 함수들(cursorHide(), gotoXY(int x, int y), textcolor(int foreground, int background)은 구현이 단순하고, 시간/공간 복잡도, 자료구조 및 알고리즘에서 특이한 점이 없으므로 설명을 생략한다.

int direction(int dir)

```
// dy[], dx[] 방향 idx 값을 방향 상수로 변환
int direction(int dir) {
    if (dir == 0) return UP;
    else if (dir == 1) return RIGHT;
    else if (dir == 2) return DOWN;
    else if (dir == 3) return LEFT;
    else return 0;
}
```

dy[], dx[] 방향 idx 값을 방향 상수로 변환한다. **시간 복잡도, 공간 복잡도는 모두 O(1)**이다.

void getFood(int x, int y)

```
// 플레이어의 아이템 획득 처리
void getFood(int x, int y) {

    // 입력받은 좌표에 음식이 있다면 음식을 먹고 점수 가산
    if (Map[y][x] == FOOD) {
        Map[y][x] = BLANK;
        score += 10;
    }

    // 입력받은 좌표에 스페셜 아이템이 있다면 스페셜 아이템을 먹고 점수 가산
    if (Map[y][x] == SPECIAL) {
        Map[y][x] = BLANK;
        SPECIAL_FLAG = 100; // 스페셜 아이템의 타이머 가산
        score += 1000;
    }
}
```

플레이어의 좌표를 입력받아 아이템 획득을 처리한다. 만약 해당 좌표에 음식이 있었다면 음식을 먹고 점수를 가산하고, 스페셜 아이템이 있다면 스페셜 아이템을 먹고 점수를 가산한다. 스페셜 아이템을 먹으면 SPECIAL_FLAG 타이머가 작동된다. 좌표 값만 살펴보므로 시**간 복잡도, 공간 복잡도는 모두 O(1)**이다.

void meetGhost(int cx, int cy, Ghost& ghost)

```
// 캐릭터 좌표, 유령 좌표를 입력받아 유령과 플레이어의 충돌 처리
void meetGhost(int cx, int cy, Ghost& ghost) {
    // 유령의 좌표와 플레이어의 좌표가 같다면
    if (cx == ghost.getX() && cy == ghost.getY()) {
        if (!SPECIAL_FLAG) game_over = 1; // 스페셜 아이템이 없을 시 게임 오버
        else ghost.die();                 // 스페셜 아이템이 있을 시 유령을 쓰러뜨림
    }
}
```

캐릭터와 유령의 좌표를 입력받아 유령과 플레이어의 충돌을 처리한다. 충돌 시 스페셜 아이템이 없다면 플레이어는 게임을 패배하게 되며, 스페셜 아이템이 있다면 유령을 쓰러뜨릴 수 있다. 유령과 플레이어의 좌표만 입력받으므로 **시간 복잡도, 공간 복잡도는 모두 O(1)**이다.

bool isMovable(int x, int y, int dir)

```
// 좌표와 이동 방향 상수를 입력받아 이동 여부를 출력
bool isMovable(int x, int y, int dir) {
    // 입력받은 이동 방향에 따라 좌표 이동
    switch (dir) {
    case UP:
        y = max(1, y - 1); break;
    case LEFT:
        x = max(1, x - 1); break;
    case RIGHT:
        x = min(23, x + 1); break;
    case DOWN:
        y = min(23, y + 1); break;
    }

    // 이동 방향이 벽이라면 이동 불가능, 벽이 아니라면 이동 가능
    if (Map[y][x] != WALL) return true;
    else return false;
}
```

플레이어의 현재 좌표와 이동 방향 상수를 입력받아 이동 가능 여부를 출력한다. 현재 좌표에 대해 다음 좌표의 이동 여부만 판정하므로 **시간, 공간 복잡도는 모두 O(1)**이다.

void printEntity(const int x, const int y, const char *string, int color)

```
// 좌표와 문자열, 문자 색을 입력받아 개체 출력
void printEntity(const int x, const int y, const char *string, int color) {
    gotoXY(2 * x + MAP_POS_X, y + MAP_POS_Y); // 좌표로 이동
    textColor(color, BLACK);                  // 입력받은 색으로 문자열 변경
    cout << string;                           // 문자열 출력
}
```

좌표와 문자열, 문자 색을 입력받아 개체를 출력한다. 좌표, 상수 문자열, 색을 입력받아 좌표에 문자열을 출력하므로 **시간, 공간 복잡도는 모두 O(1)**이다.

void moveEntity(int& x, int& y, int dir, const char *string, int color)

```
// 좌표와 방향 상수를 입력받아 이동 여부를 판단한 뒤 이동 처리
void moveEntity(int& x, int& y, int dir, const char *string, int color) {
    if (isMovable(x, y, dir)) {
        printEntity(x, y, "  ", 0); // 기존 좌표에 출력된 값은 삭제

        // 적절히 좌표 이동
        switch (dir) {
        case UP:
            y = max(1, y - 1); break;
        case LEFT:
            x = max(1, x - 1); break;
        case RIGHT:
            x = min(23, x + 1); break;
        case DOWN:
            y = min(23, y + 1); break;
        }

        printEntity(x, y, string, color); // 이동한 좌표에서 새로 개체 출력
    }
}
```

좌표와 문자열, 방향 상수, 상수 문자열, 문자 색을 입력받아 현재 좌표에 출력된 값을 삭제하고, 이동한 좌표에 값을 출력한다. 입력받은 1개의 좌표에 따라 문자열을 출력하는 작업만 수행하므로 **시간, 공간 복잡도는 모두 O(1)**이다.

void dfs(int x, int y)

```
// 미로 생성을 위한 recursive-dfs 알고리즘
void dfs(int x, int y) {
    vector<int> dir = { 0, 1, 2, 3 }; // 방향 idx 1차원 벡터 생성
    shuffle(dir.begin(), dir.end(), gen); // 난수 변수를 바탕으로 shuffle해 무작위화
    Map[y][x] = BLANK, vst[y][x] = 1; // 시작 좌표를 비우고, 방문 처리함

    // DFS 처리
    for (auto i : dir) { // 무작위 방향 idx를 dir 벡터에서 꺼낸 뒤
        int nx = x + 2 * dx[i], ny = y + 2 * dy[i]; // 해당 방향으로 2칸 이동
        if (nx <= 0 || ny <= 0 || nx >= 24 || ny >= 24) continue; // 경계값 처리

        // 방문하지 않은 노드라면 중간의 벽을 부수고 dfs(nx, ny) 재귀적 실행
        if (!vst[ny][nx]) {
            Map[y + dy[i]][x + dx[i]] = BLANK;
            dfs(nx, ny);
        }
    }
}
```

미로 생성을 수행하는 recursive-dfs 알고리즘이다. 다음과 같은 동작으로 구성된다.

1) 방향 idx로 구성된 1차원 벡터 dir을 생성하고, 난수를 이용해 무작위로 섞는다.
2) 시작 좌표를 비우고 방문 처리한다.
3) dir에서 뽑은 방향대로 2칸 이동하고, 경계를 벗어났는지/이미 방문했는지 확인한다.
4) 맵 안에 있는 방문하지 않은 노드라면 벽을 부수고 dfs(nx, ny)를 재귀적으로 실행한다.

dfs 알고리즘을 통해 맵을 생성할 때까지 맵의 **정점 V개에 대해** 4가지 방향을 모두 확인해야 하며, 중복된 정점을 탐색하는 경우는 존재하지 않으므로(vst 방문 처리) **O(4 * V)의 시간 복잡도, 즉 O(V)**의 시간 복잡도 가지고 있다고 볼 수 있다. 이는 **미로가 매우 sparse한 그래프이기 때문에 가능하다.**

시간 복잡도를 O(V + E)로도 표현할 수 있는데. 이 경우는 정점들 사이의 벽을 간선이라고 가정하는 경우이다. 실제 맵은 25 * 25 사이즈로 구성되어 있으며 x좌표, y좌표가 모두 홀수인 경우만 정점으로 취급되므로, 정점을 대략적으로 계산하면 144개 존재한다고 볼 수 있다.

맵을 전체적으로 탐색하고, 방문 배열을 따로 저장하므로 **공간 복잡도는 O(2 * V), O(V)**로 고려할 수 있다. 시간 복잡도와 공간 복잡도를 맵의 기준에서 살펴본다면, 시간 복잡도를 O(WIDTH * HEIGHT * 4), 공간 복잡도를 O(2 * WIDTH * HEIGHT)로 생각해볼 수도 있다.

void initPacMan()

```
// 게임 초기화 및 미로 생성 함수
void initPacMan() {
    cursorHide(); // 커서 깜빡임 삭제

    // 지도에 벽 생성
    for (int y = 0; y < 25; y++) {
        for (int x = 0; x < 25; x++) {
            Map[y][x] = WALL;
        }
    }

    // recursive-dfs 방식을 통해 미로 랜덤 생성
    dfs(1, 1);

    // 게임의 편의성을 위해 미로를 불완전 미로로 변형
    for (int y = 1; y < 24; y++) {
        for (int x = 1; x < 24; x++) {
            // 맵의 테두리를 전부 비움
            if (x == 1 || y == 1 || x == 23 || y == 23)
                Map[y][x] = BLANK;

            // x % 2 == 1, y % 2 == 1의 맵에 있는 벽을 25% 확률로 파괴
            if (x % 2 || y % 2) {
                if (dis(gen) == 1) Map[y][x] = BLANK;
            }
        }
    }

    // 먹어야 하는 아이템 생성
    for (int y = 1; y < 24; y++) {
        for (int x = 1; x < 24; x++) {
            if (!Map[y][x]) Map[y][x] = FOOD;
        }
    }

    // 스페셜 푸드 생성
    Map[23][23] = SPECIAL;
}
```

cursorHide() 유틸리티 함수로 커서를 숨기고, dfs(1, 1)로 미로를 생성하고, 미로를 불완전 미로로 변형한 뒤 아이템, 스페셜 아이템을 생성한다. dfs() 함수에서 미로 생성의 시간 복잡도는 이미 살펴봤으므로 이를 제외하면, 벽을 생성하고, 미로를 불완전 미로로 변형하며, 아이템을 생성하는 과정에서 **맵을 3번 탐색하므로 시간 복잡도는 O(3 * WIDTH * HEIGHT)로 O(WIDTH * HEIGHT), 공간 복잡도는 O(WIDTH * HEIGHT)**이다.

void printUI()

```
// UI 출력 함수
void printUI() {
    // 점수 UI 출력
    for (int y = 20; y <= 24; y++) {
        for (int x = 27; x <= 37; x++) {
            if (x > 27 && x < 37 && y > 20 && y < 24) continue;
            printEntity(x, y, "▨", 15);
        }
    }

    // 점수 출력
    gotoXY(58 + MAP_POS_X, 22 + MAP_POS_Y);
    cout << "SCORE: " << setw(7) << score;

    // 라운드 UI 출력
    for (int y = 0; y <= 4; y++) {
        for (int x = 27; x <= 37; x++) {
            if (x > 27 && x < 37 && y > 0 && y < 4) continue;
            printEntity(x, y, "▨", 15);
        }
    }

    // 라운드 출력
    gotoXY(58 + MAP_POS_X, 2 + MAP_POS_Y);
    cout << "ROUND: " << setw(7) << playing_round;

    // 스페셜 아이템 획득 시
    if (SPECIAL_FLAG > 0) {
        gotoXY(54 + MAP_POS_X, 6 + MAP_POS_Y);
        textColor(YELLOW, BLACK);       // 글자색 변경
        cout << "GOT SPECIAL ITEM !!!"; // 스페셜 아이템 획득 메시지 출력
        textColor(WHITE, BLACK);                    // 글자색 초기화
    }
}
```

UI를 출력하는 함수로, 점수 UI, 라운드 UI, 스페셜 아이템 획득 메시지를 출력한다. **시간 복잡도는 O(UI_WIDTH * UI_HEIGHT), 공간 복잡도는 따로 값을 저장하지 않으므로 O(1)**이다. 시간 복잡도의 UI_WIDTH, UI_HEIGHT 값은 그리 크지 않으므로 상수로 취급해도 무방하다.

void printMap(int cx, int cy, Ghost ghost1, Ghost ghost2)

```
// 맵 출력
void printMap(int cx, int cy, Ghost ghost1, Ghost ghost2) {
    textColor(WHITE, BLACK);
    for (int y = 0; y < 25; y++) {
        for (int x = 0; x < 25; x++) {

            // 게임의 깜빡임을 줄이기 위해 캐릭터의 좌표, 유령의 좌표일 경우 skip
            if (y == cy && x == cx) continue;
            if (y == ghost1.getY() && x == ghost1.getX()) continue;
            if (y == ghost2.getY() && x == ghost2.getX()) continue;

            if (Map[y][x] == WALL) printEntity(x, y, "▦", WHITE);           // 벽 출력
            if (Map[y][x] == FOOD) printEntity(x, y, "·", YELLOW);          // 음식 출력
            if (Map[y][x] == SPECIAL) printEntity(x, y, "★", LIGHTMAGENTA); // 스페셜 아이템 출력
        }
    }

    // 프로젝트 메시지 출력
    gotoXY(MAP_POS_X, 26 + MAP_POS_Y); cout << "20191150 전현길: PACMAN PROJECT";
}
```

맵을 순회하며 현재 상태를 출력한다. 게임의 깜빡임을 줄이기 위해 캐릭터의 좌표, 유령의 좌표일 경우 맵을 출력하지 않는다(=어차피 개체를 출력할 때 가려질 것이기 때문). 해당 조건문을 사용함으로써 반복문 연산이 복잡해지지만, 더블 버퍼링 등의 기능을 따로 구현할 필요 없이 깜빡임 없이 부드럽게 게임을 진행할 수 있었다. 이외엔 맵의 좌표 정보에 따라 적절한 문자열을 출력한다. **시간, 공간 복잡도는 모두 맵의 크기에 맞춰 O(WIDTH * HEIGHT)**이다. **전역 변수로 맵을 사용하기 때문에 공간 복잡도를 O(1)이라고 볼 수도 있다.**

void isGameOver()

```
void isGameOver() {
    // 게임 종료 여부 확인
    int flag = 0; // flag reset
    for (int y = 0; y < 25; y++) {
        for (int x = 0; x < 25; x++) {
            if (Map[y][x] == FOOD) flag = 1; // 맵에 음식이 있을 시 flag set
        }
    }

    if (!flag) game_win = 1; // 맵에 음식이 없다면 게임 승리 판정
}
```

맵을 순회하며 맵에 음식이 더 이상 없다면 게임 승리를 판정한다. 맵을 전부 순회하므로 시간, 공간 복잡도는 O(WIDTH * HEIGHT)이다. **전역 변수로 맵을 사용하기 때문에 공간 복잡도를 O(1)이라고 볼 수도 있다.**

void win(), void lose()

```
// 승리에 따른 메시지 출력
void win() {
    textColor(YELLOW, BLACK);

    gotoXY(8, 9);  cout << "▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦";
    gotoXY(8, 10); cout << "▦                                                          ▦";
    gotoXY(8, 11); cout << "▦    ▨▨      ▨▨  ▨▨▨▨▨  ▨▨    ▨▨  ▨  ▨  ▨    ▦";
    gotoXY(8, 12); cout << "▦    ▨▨      ▨▨    ▨▨      ▨▨▨  ▨▨  ▨  ▨  ▨    ▦";
    gotoXY(8, 13); cout << "▦    ▨▨      ▨▨    ▨▨      ▨▨▨▨▨▨  ▨  ▨  ▨    ▦";
    gotoXY(8, 14); cout << "▦    ▨▨  ▨  ▨▨    ▨▨      ▨▨  ▨▨▨                ▦";
    gotoXY(8, 15); cout << "▦      ▨▨  ▨▨    ▨▨▨▨▨  ▨▨    ▨▨  ▨  ▨  ▨    ▦";
    gotoXY(8, 16); cout << "▦                                                          ▦";
    gotoXY(8, 17); cout << "▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦";
    gotoXY(-1, -1);

    Sleep(5000); // 메시지 출력 시간
}

// 패배에 따른 메시지 출력
void lose() {
    textColor(LIGHTRED, BLACK);

    gotoXY(6, 9);  cout << "▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦";
    gotoXY(6, 10); cout << "▦                                                                  ▦";
    gotoXY(6, 11); cout << "▦    ▨▨          ▨▨▨    ▨▨▨▨▨  ▨▨▨▨▨                ▦";
    gotoXY(6, 12); cout << "▦    ▨▨        ▨▨  ▨▨  ▨▨        ▨▨                      ▦";
    gotoXY(6, 13); cout << "▦    ▨▨        ▨▨  ▨▨  ▨▨▨▨▨  ▨▨▨▨▨                ▦";
    gotoXY(6, 14); cout << "▦    ▨▨        ▨▨  ▨▨        ▨▨  ▨▨                      ▦";
    gotoXY(6, 15); cout << "▦    ▨▨▨▨▨    ▨▨▨    ▨▨▨▨▨  ▨▨▨▨▨  ▨  ▨  ▨    ▦";
    gotoXY(6, 16); cout << "▦                                                                  ▦";
    gotoXY(6, 17); cout << "▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦▦";
    gotoXY(76, 25);

  Sleep(5000); // 메시지 출력 시간
}
```

승리, 패배 조건에 따라 적절한 메시지를 출력한다. 시간, 공간 복잡도는 O(1)이다.

```
Ghost::Ghost(int pos_x, int pos_y, int pos_dir, int color);
void Ghost::print() const
int Ghost::getY() const
int Ghost::updateTime()
int Ghost::getDead()
void Ghost::die()
```

```
// 현재 좌표, 방향 상수, 색깔을 입력받는 유령 개체 생성자
Ghost(int pos_x, int pos_y, int pos_dir, int color) {
    x = pos_x;
    y = pos_y;
    dir = pos_dir;
    this->color = color;
    origin_color = color; // 생성될 때의 색을 본래색으로 저장
    time = 0;             // 타이머 초기화
    isDead = 0;           // 생존 여부 초기화
}
void print() const { printEntity(x, y, "●", color); } // 유령의 현재 좌표에서 값 출력
int getX() const { return x; }          // x 좌표 getter
int getY() const { return y; }          // y 좌표 getter
int updateTime() { return time--; }     // 타이머 계산 및 현재 타이머 반환
int getDead() const { return isDead; } // 생존 여부 getter

// 스페셜 아이템을 획득한 캐릭터와 충돌 시 유령 사망 판정
void die() {
    x = 23, y = 23;
    isDead = 50;
}
```

Ghost 함수의 단순한 getter 함수, 출력 함수, 생성자 함수들이다. die() 함수는 유령의 사망 판정을 수행하는 함수이다. **시간 복잡도, 공간 복잡도는 전부 O(1)**이다. getter 함수에서는 값을 변화시키지 않기 때문에 const 함수로 저장했다.

void Ghost::update(int character_x, int character_y)

```
// 유령의 상태 정보 갱신
void update(int character_x, int character_y) {
    time = 2; // 타이머 초기화

    // 죽어 있는 상태라면 살아날 때까지 길찾기 정지
    if (isDead-- > 0) {
        x = 23, y = 23;
        color = LIGHTGRAY;
        print();
        return;
    }
    else if (SPECIAL_FLAG) color = BLUE; // 스페셜 음식을 먹었다면 색깔 변경
    else color = origin_color;           // 그렇지 않다면 본래색으로 변경

    move(character_x, character_y);      // 캐릭터 이동
}
```

유령의 상태 정보를 갱신하는 함수이다. 실행될 때마다 타이머를 초기화하여 유령의 상태 갱신 주기를 결정한다. 또한 죽어 있는 상태인지, 플레이어가 스페셜 음식을 먹은 상태인지에 따라서 색깔을 변화시킨다. **시간 복잡도, 공간 복잡도는 O(1)**이다.

void Ghost::move(int character_x, int character_y)

```
// 길찾기를 통한 주인공 캐릭터 팩맨 탐색
void move(int character_x, int character_y) {
    int vst[25][25]; memset(vst, 0x3f, sizeof(vst));
    queue<Path> q; vector<Vertex> initial_path;

    // 주변에 캐릭터가 있는지 BFS 탐색, 캐릭터의 위치에 따라 dir 갱신
    q.push(Path(x, y, 0, initial_path));
    vst[y][x] = 1;

    // 표준적인 BFS 알고리즘에서 경로만 추가적으로 저장함
    while (q.size()) {
        int cx, cy, cdst; vector<Vertex> path;
        q.front().copy(cx, cy, path); // 좌표와 경로 변수에 저장
        cdst = vst[cy][cx];
        q.pop(); // 큐 pop

        // 캐릭터를 탐색했을 경우, 캐릭터를 향한 최단 경로로 이동
        if (cx == character_x && cy == character_y && path.size() > 1) {
            dir = path[1].dir; // 방향 정보 저장
            break;
        }

        if (path.size() > 50) break; // 맵 전체를 커버할 수 있도록 설정

        for (int i = 0; i < 4; i++) {
            int ny = cy + dy[i], nx = cx + dx[i], ndst = cdst + 1;    // 방향 idx에 따른 다음 좌표 계산
            if (nx <= 0 || ny <= 0 || nx >= 24 || ny >= 24) continue; // 경계값 처리
            if (Map[ny][nx] == WALL || vst[ny][nx] < ndst) continue;  // 이동 불가능하거나, 최단 거리가 아닐 시 통과
            vst[ny][nx] = ndst;                                       // 최단 거리일 시 방문 배열에 저장
            q.push(Path(nx, ny, direction(i), path)); // Path 큐에 push
        }
    }

    moveEntity(x, y, dir, "●", color); // 최단 경로를 향해 이동
}
```

캐릭터의 좌표와 유령의 좌표를 바탕으로, **큐 자료구조와 함께 BFS 알고리즘을 수행**하여 캐릭터를 향한 최단 경로를 탐색한다. **시간 복잡도는 최악의 경우 맵을 전부 탐색하므로 O(WIDTH * HEIGHT)**이며, **공간 복잡도는 최악의 경우 맵의 모든 정점이 저장되고, 각 정점이 가질 수 있는 최대 경로는 맵의 크기만큼이므로 O(WIDTH^2 * HEIGHT^2)**이다. 단, 큐에 저장된 정점의 개수와 큐에 저장된 경로의 길이에는 서로 반비례 관계가 성립하므로 **실제로는 이보다 훨씬 적은 공간 복잡도를 갖게 된다.**

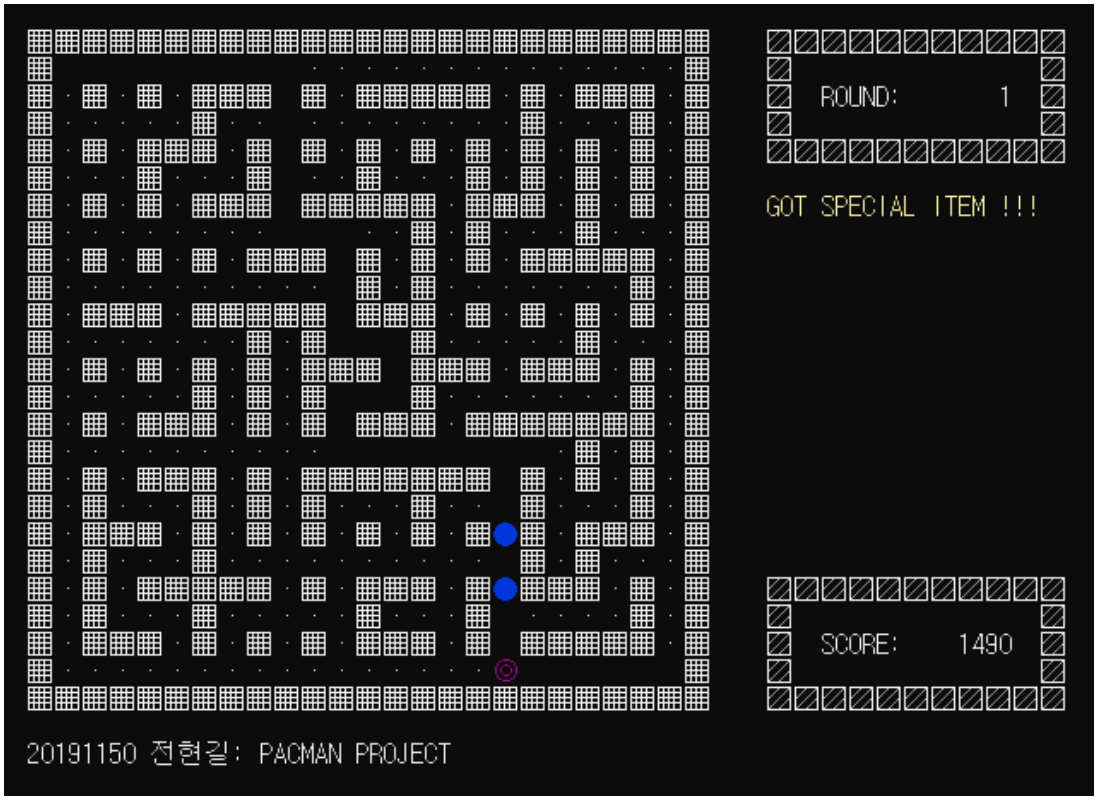
6. 프로젝트 실행 결과 이미지 및 창의적 구현 항목에 대한 설명

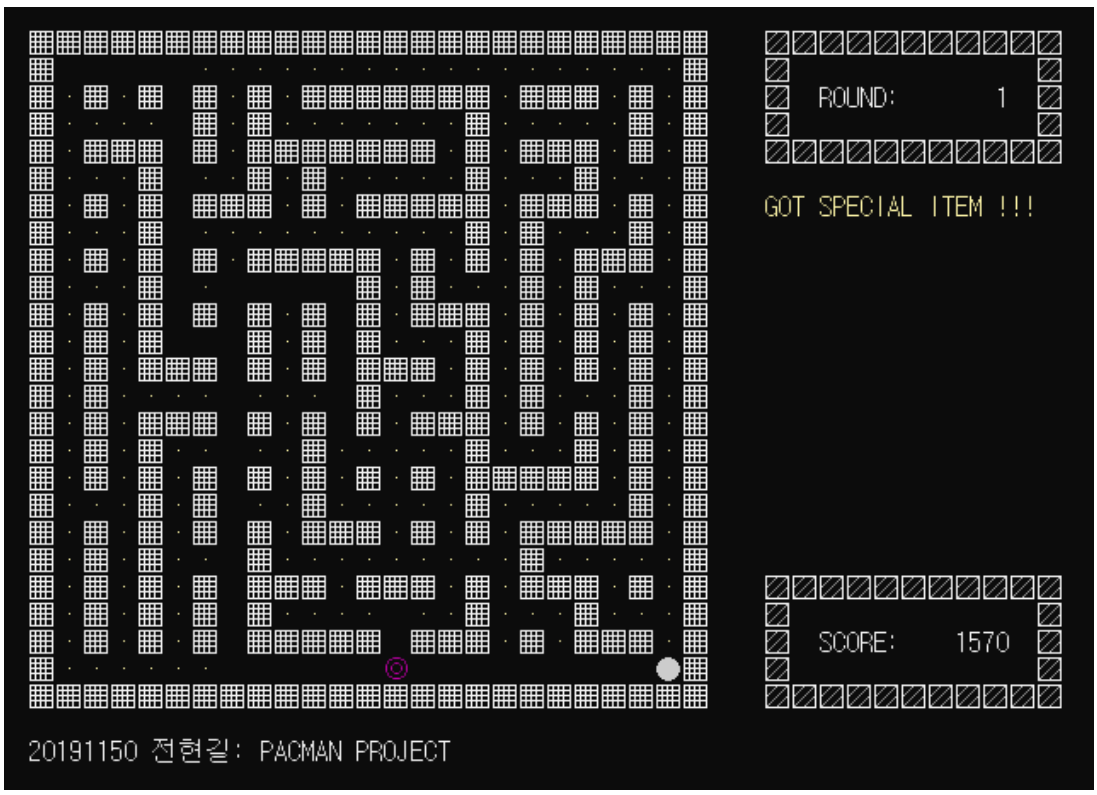
플레이 화면

노란색 ◎ 아이콘은 플레이어 캐릭터인 팩맨을 표시하며, 두 붉은색, 하늘색 ● 아이콘은 적 캐릭터인 유령을 표시한다. ▦ 아이콘은 벽을 표현하며, 자주색 ★ 아이콘은 스페셜 아이템을 의미한다.

플레이어는 유령을 피해서 · 아이콘으로 표현된 모든 아이템과 스페셜 아이템을 획득하면 승리하며, 그 전에 유령과 충돌하여 사망하면 패배한다. 스페셜 아이템을 획득하면 플레이어는 잠시 동안 무적 상태가 되며, 무적 상태에서 유령과 충돌하면 오히려 유령을 쓰러뜨릴 수 있다. 죽은 유령은 맵의 오른쪽 아래 스페셜 아이템이 있는 위치에서 일정 시간 후에 부활한다.

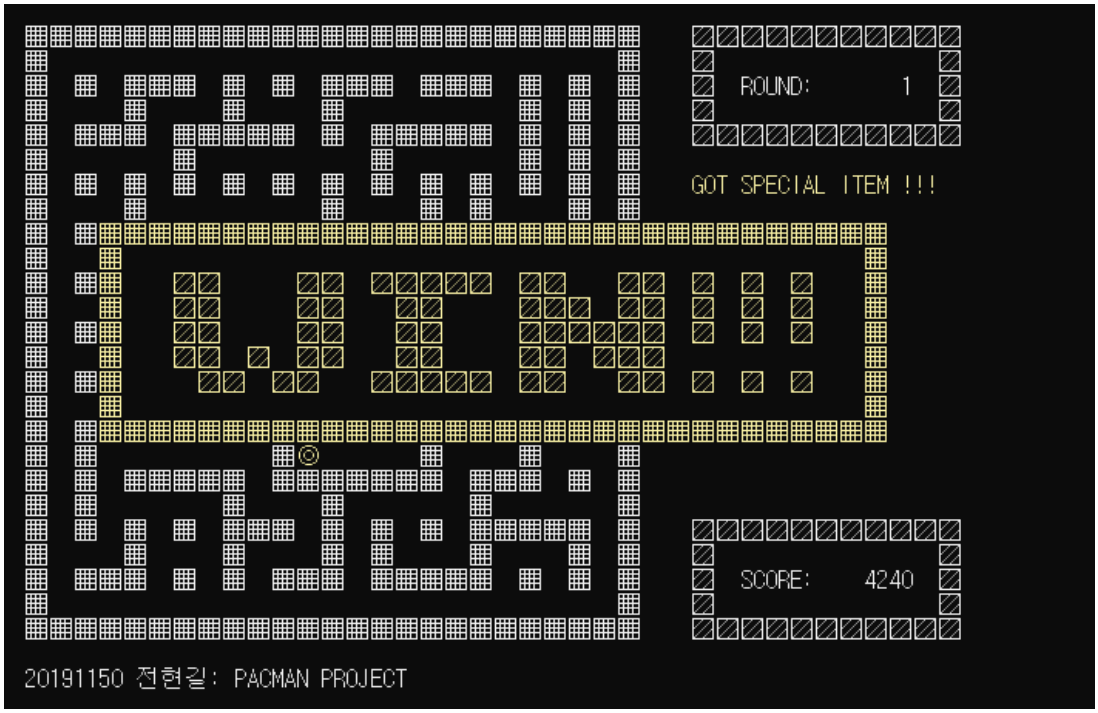
스페셜 아이템 획득 시 화면



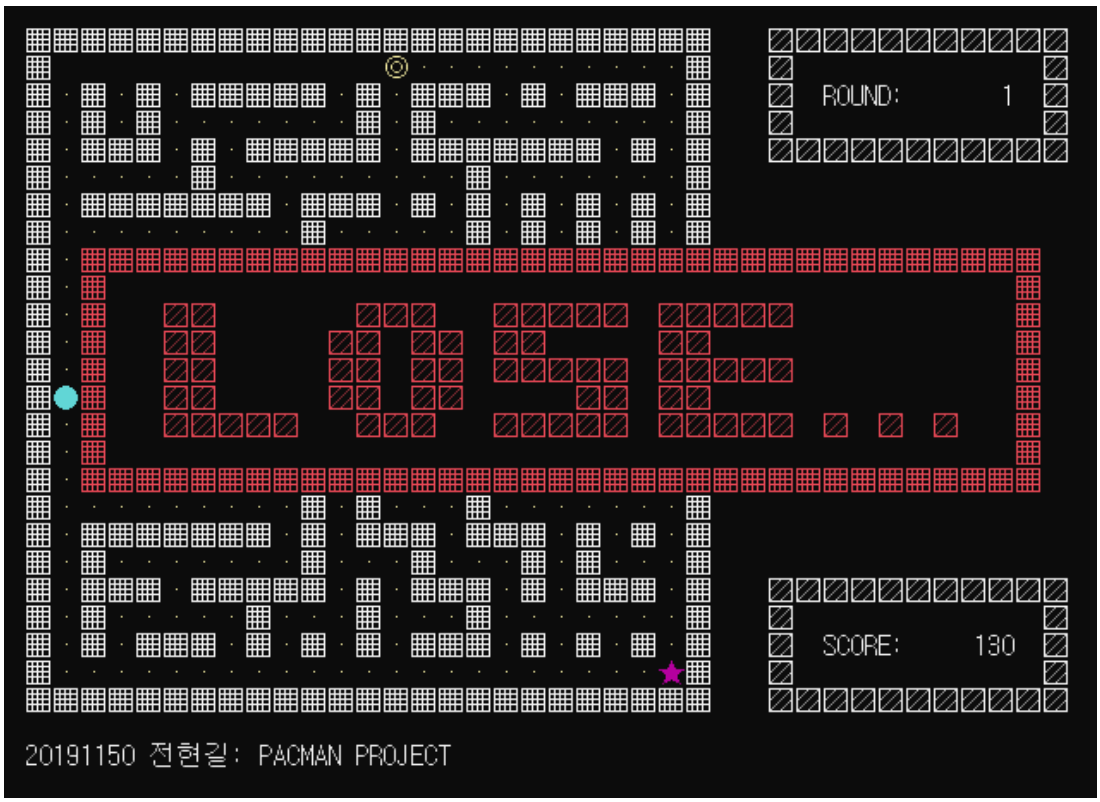


스페셜 아이템을 획득하면 플레이어 캐릭터의 색은 자주색으로 바뀌고, 유령은 파란색으로 바뀐다. 이 상태에서 유령과 충돌해 유령을 쓰러뜨리면 오른쪽 아래처럼 유령이 회색으로 깜빡이다가 부활한다.

승리 화면


ROUND: 1
GOT SPECIAL ITEM !!!
SCORE: 4240
20191150 전현길: PACMAN PROJECT

패배 화면


ROUND: 1
SCORE: 130
20191150 전현길: PACMAN PROJECT

7. 느낀 점 및 개선 사항

수업을 들으면서 실제로 조작할 수 있는 게임을 내 손으로 바닥부터 직접 만들어본 것은 이번이 처음이었다. 테트리스 프로젝트에서도 유사하게 조작 가능한 게임을 만들어 보았지만, 스켈레톤 코드에서 시작해 살을 붙였던 것이었기 때문에 성취감이 크지 않았는데, 이번 팩맨 프로젝트의 경우에는 처음부터 끝까지 직접 만들어 보았기 때문에 성취감이 컸다.

또 게임에서 어떻게 사용자의 입력을 받는지, 화면을 어떻게 출력하는지, 어떻게 해야 사용자 입력을 받지 않아도 시간이 흐르게 만들 수 있는지 등에 대하여 아주 개략적인 이해만 갖고 있었는데, 이번 기회에 직접 게임을 구현하면서 해당 부분들에 대해서도 직관적으로 이해할 수 있게 되었다.

다만 화면을 부드럽게 출력하기 위한 여러 테크닉, 특히 더블 버퍼링을 처음부터 염두에 두고 게임을 구현했다면 좋았을 텐데, 프로젝트를 처음 시작할 때는 이런 정보를 몰랐기 때문에 구현하지 못한 것은 아쉽다. 더블 버퍼링을 구현하지 못한 결과로 printMap()에서 너무나 많은 조건문을 처리할 수밖에 없었고, 게임플레이에 지장을 줄 정도는 아니지만 조금의 버벅임이 생긴 것을 체감할 수 있었다.

다음으로 아쉬운 점은 프로젝트에서 추상화를 충분히 수행하지 못했다는 점다. 팩맨과 유령을 각각 클래스로 저장한 뒤, 팩맨과 유령을 추상화한 상위 클래스인 Entity를 만들어 중복되는 정보(좌표, 방향, 생존 여부 등)나 멤버 함수를 저장할 수 있게 하면 코드 구조가 훨씬 더 깔끔해졌을 듯하다.

main() 함수도 좀 더 깔끔하게 쓰기 위해서 유령 각각에 대해 여러 가지 상태 변경 함수들(meetGhost(), ghost1.update())을 다 그대로 쓰기보다는 판정을 하기보단 gameUpdate() 함수 등의 상위 함수들을 작성했다면 좋았을 것 같다. 특별히 사용자 입력을 기다리지 않아도 되는 함수들의 경우 모두 작성하니 main() 함수가 조금 읽기 불편해진다고 느꼈다.

생각해본 개선 사항들은 다음과 같다. 그래픽 라이브러리를 통한 벽, 팩맨, 유령, 아이템 등의 그래픽 개선, 사운드 라이브러리를 통한 효과음/배경음악 재생, 원본 게임에도 존재하는 화면 좌우로 이동하는 팩맨의 텔레포트 구현(유령의 BFS 길찾기 함수의 수정이 필요), 점수를 바탕으로 테트리스 프로젝트에서 수행했던 것과 유사한 랭킹 시스템 구현, 라운드 종료 시 걸린 시간에 따라 별점 평가 시스템 도입, 아이템을 소지했다가 원할 때 사용할 수 있도록 기능 추가 등이다. 게임의 기본 틀만 잡아 게임이 상당히 심심하므로 여러 기믹들을 더 추가할 수 있으면 좋겠다.