**C++프로그래밍PROJECT**

박지선

이효준

최광희

**[1단계]**

**< Game.h 파일 >**

class Game

{

private:

int currentStage; // 현재 스테이지

vector<vector<vector<int>>> maps; // 맵을 담는 3차원 벡터

public:

int getCurrentStage() { return currentStage; }

vector<vector<vector<int>>> getMaps() { return maps; }

void gameStart(); // 게임 시작

void setMaps(); // 맵을 세팅

};

**< Game.cpp 파일 >**

// 게임을 시작하여 맵을 세팅

void Game::gameStart()

{

setMaps();

}

void Game::setMaps()

{

int numOfRows, numOfCols; // 맵의 행, 열의 수

ifstream is("maps.txt"); // “maps.txt” 파일

is >> numOfStages; // 총 스테이지 수

this->maps.assign(numOfStages, vector<vector<int>>()); // 2차원 벡터 초기화

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(1) 총 스테이지 수 만큼

(2) 각 map의 행과 열의 수를 입력 받아

(3) 벡터를 -1로 초기화한 후,

(4) “maps.txt” 파일로부터 맵 정보를 입력받는다.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int i = 0; i < numOfStages; i++)

{

is >> numOfRows >> numOfCols;

this->maps[i].assign(numOfRows, vector<int>(numOfCols, -1));

for (int j = 0; j < numOfRows; j++)

for (int k = 0; k < numOfCols; k++)

is >> this->maps[i][j][k];

}

}

**< main.cpp파일 >**

int main()

{

initscr(); // Curses 모드 시작, 기본 크기의 윈도우 생성

resize\_term(40, 100); // terminal크기 조정, window까지 동시 조정, 40줄, 100칸

keypad(stdscr, TRUE); // 입력 시 키보드 특수 키 입력을 가능하게 설정 // stdscr은 default window포인터, TRUE는 사용 가능

curs\_set(0); // 화면에 보이는 커서 설정, 0은 커서 사라짐

noecho(); // 문자 입력 시 입력한 값을 화면에 보이지 않게 함

start\_color(); // Color 사용 선언, 성공시OK(0), 에러시ERR(-1)

init\_pair(1, COLOR\_YELLOW, COLOR\_BLACK); // 색 attribute설정

// (팔레트, 폰트 색 , 폰트 배경색)

bkgd(COLOR\_PAIR(1)); // 한 attribute로 윈도우 전체 적용

attron(COLOR\_PAIR(1)); // Attribute적용, 1번 팔레트 사용

Game game; // Game 클래스 객체 생성

game.gameStart(); // 게임 시작

border('\*', '\*', '\*', '\*', '\*', '\*', '\*', '\*'); // \*로 윈도우 경계선 설정

// 맵 그리기

for (int j = 0; j < game.getMaps()[game.getCurrentStage()].size(); j++)

{

for (int k = 0; k < game.getMaps()[game.getCurrentStage()][j].size(); k++)

// 만약 값이 4이면 공백으로 출력, 아니면 원래 값대로 출력

if (game.getMaps()[game.getCurrentStage()][j][k] == 4)

mvprintw(20 + j, 20 + k, " ");

else

myprintw(

20 + j, 20 + k, “%d”,

game.getMaps()[game.getCurrentStage()][j][k]

);

}

}

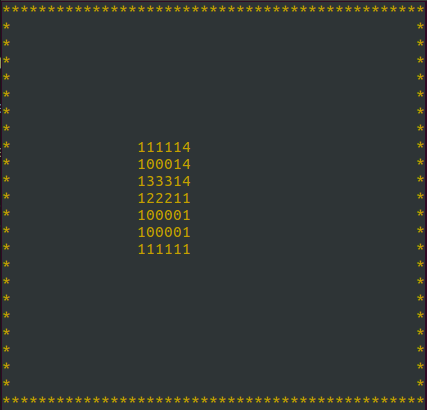
refresh(); // 디폴트 윈도우 내용을 실제 스크린에 출력

attroff(COLOR\_PAIR(1)); // Attribute 해제, 1번 팔레트 사용 해제

endwin(); // Curses 모드 종료

}

**< 실행 화면 >**



**[2단계]**

**< Game.h 파일 >**

class Game

{

private:

int currentStage;

vector<vector<vector<int>>> maps;

vector<pair<int, int>> playerPositions; // 플레이어 위치 벡터 추가

public:

int getCurrentStage() { return currentStage; }

vector<vector<vector<int>>> getMaps() { return maps; }

void gameStart();

void setMaps();

void setPlayer(); // 맵에 플레이어를 위치시킴

void move(const int direction); // 움직임

// walk(): 플레이어 앞에 상자가 없을 때 앞으로 감

// push(): 플레이어 앞에 상자가 있을 때 상자를 밀면서 앞으로 감

void walk(const int y, const int x, const int nextY, const int nextX);

void push( const int y, const int x,

const int nextY, const int nextX,

const int afterY, const int afterX);

bool checkGoal(const int y, const int x) const; // 해당 위치가 목적지인지 체크

};

**< Game.cpp 파일 >**

void Game::gameStart()

{

setMaps();

setPlayer(); // 맵에 플레이어를 위치시킴

}

// 대부분 1단계와 같고 목적지 위치 저장 기능이 추가되었다.

void Game::setMaps()

{

/\*

생략

\*/

for (int i = 0; i < numOfStages; i++)

{

is >> numOfRows >> numOfCols;

this->maps[i].assign(numOfRows, vector<int>(numOfCols, -1));

for (int j = 0; j < numOfRows; j++)

{

for (int k = 0; k < numOfCols; k++)

{

is >> this→maps[i][j][k];

// 해당 위치가 목적지일 경우 목적지 벡터에 추가함

if (maps[i][j][k] == 3)

goalPositions[i].push\_back(make\_pair(j, k));

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(1) “PlayerPosition.txt” 파일에 있는 플레이어 위치를 입력 받아

(2) playerPositions 벡터에 push 한 후,

(3) 맵에 위치 시킨다.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Game::setPlayer()

{

int y, x;

ifstream is("PlayerPositions.txt");

is >> numOfStages;

for (int i = 0; i < numOfStages; i++)

{

is >> y >> x;

playerPositions.push\_back(make\_pair(y, x));

}

for (int i = 0; i < numOfStages; i++)

maps[i][playerPositions[i].first][playerPositions[i].second] = 5;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

캐릭터를 입력된 방향으로 움직이는 함수

direction: KEY\_UP or KEY\_DOWN or KEY\_LEFT or KEY\_RIGHT

y, x: 현재 위치

nextY, nextX: 입력된 방향에 해당하는 다음 위치

afterY, afterY: 그 다음 위치

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Game::move(const int direction)

{

int y = playerPositions[currentStage].first;

int x = playerPositions[currentStage].second;

int nextY, nextX, afterY, afterX;

//입력이 방향키 up일 경우

if (direction == KEY\_UP) { nextY = y – 1; nextX = x;

afterY = y – 2; afterX = x;}

//입력이 방향키 down일 경우

else if (direction == KEY\_DOWN) { nextY = y + 1; nextX = x;

afterY = y + 2; afterX = x;}

//입력이 방향키 left일 경우

else if (direction == KEY\_LEFT) { nextY = y; nextX = x - 1;

afterY = y; afterX = x – 2;}

//입력이 방향키 right일 경우

else if (direction == KEY\_RIGHT) { nextY = y; nextX = x + 1;

afterY = y; afterX = x + 2;}

// 다음칸이 벽이 아닐 때

if (maps[currentStage][nextY][nextX] != 1)

{

// 다음칸이 비어있을 때 walk()

if (maps[currentStage][nextY][nextX] == 0 ||

maps[currentStage][nextY][nextX] == 3)

walk(y, x, nextY, nextX);

// 다음칸이 상자일 때 push()

else

// 다음 다음 칸이 빈 칸일 때 push()

if (maps[currentStage][afterY][afterX] == 0 ||

maps[currentStage][afterY][afterX] == 3)

push(y, x, nextY, nextX, afterY, afterX);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

빈칸으로 한 칸 앞으로 가는 함수

(1) 플레이어를 다음칸으로 위치

(2) 기존 위치를 0으로

(3) 현재 위치를 저장하는 playerPosition 벡터 업데이트

(4) 이동 전 위치가 목적지일 경우 빈칸(0)이 아닌 목적지(3)으로 저장

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Game::walk(const int y, const int x, const int nextY, const int nextX)

{

maps[currentStage][nextY][nextX] = 5;

maps[currentStage][y][x] = 0;

playerPositions[currentStage].first = nextY;

playerPositions[currentStage].second = nextX;

if (checkGoal(y, x))

maps[currentStage][y][x] = 3;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

상자를 밀면서 한 칸 앞으로 가는 함수

(1) 다음 다음 칸을 2로 저장 (상자를 밈)

(2) 다음 칸을 5로 저장 (플레이어 이동)

(3) 기존 위치를 0으로

(4) 현재 위치를 저장하는 playerPosition 벡터 업데이트

(5) 이동 전 위치가 목적지일 경우 빈칸(0)이 아닌 목적지(3)으로 저장

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Game::push( const int y, const int x,

const int nextY, const int nextX,

const int afterY, const int afterX)

{

maps[currentStage][afterY][afterX] = 2;

maps[currentStage][nextY][nextX] = 5;

maps[currentStage][y][x] = 0;

playerPositions[currentStage].first = nextY;

playerPositions[currentStage].second = nextX;

if (checkGoal(y, x))

maps[currentStage][y][x] = 3;

}

// 입력받은 위치를 goalPosition 벡터와 비교하여 목적지인지 체크

bool Game::checkGoal(const int y, const int x) const

{

for (int i = 0; i < goalPositions[currentStage].size(); i++)

if (goalPositions[currentStage][i].first == y &&

goalPositions[currentStage][i].second == x)

return true;

return false;

}

**< main.cpp파일 >**

int main()

{

/\*

attribute 설정 생략

\*/

Game game;

game.gameStart();

while (true)

{

/\*

맵 그리기 생략

\*/

refresh();

// 입력 받아 움직이기

int input = getch();

game.move(input);

}

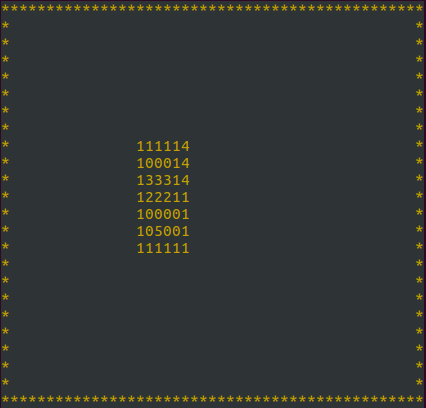
attroff(COLOR\_PAIR(1));

endwin();

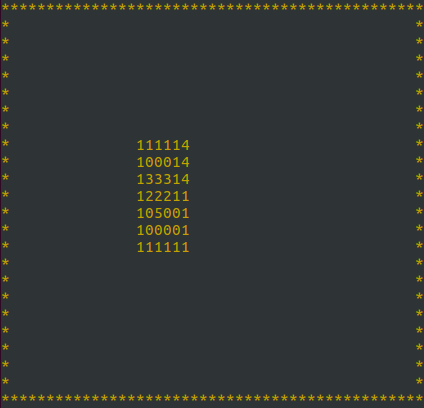
}

**< 실행 화면 >**

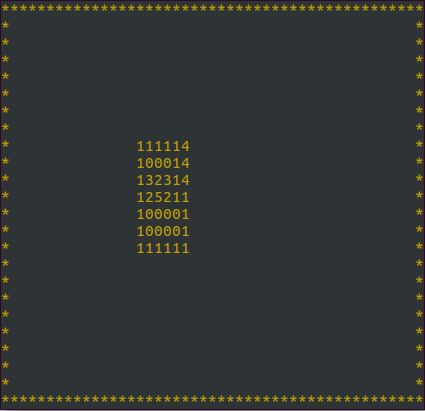
**1. 캐릭터 표시**



**2. 캐릭터 이동**



**3. 상자 밀기**



**[3단계]**

**< Game.h 파일 >**

class Game

{

private:

int numOfStages, currentStage, stepCount, pushCount; // 카운트 변수 추가

vector<bool> finished; // 스테이지 별 종료 체크

vector<vector<vector<int>>> maps;

vector<pair<int, int>> playerPositions;

vector<vector<pair<int, int>>> goalPositions;

public:

int getStepCount() { return stepCount; }

int getPushCount() { return pushCount; }

int getCurrentStage() { return currentStage; }

vector<bool> getFinised() { return finished; }

vector<vector<vector<int>>> getMaps() { return maps; }

void gameStart();

void setMaps();

void setPlayer();

void move(const int direction);

void walk(const int y, const int x, const int nextY, const int nextX);

void push(const int y, const int x, const int nextY, const int nextX, const int afterY, const int afterX);

bool checkGoal(const int y, const int x) const;

void nextStage(); // 다음 스테이지로 이동

bool checkSuccess() const; // 현재 스테이지 종료 체크

bool checkAllSuccess() const; // 모든 스테이지 종료 체크

};

**< Game.cpp 파일 >**

// 변수 및 벡터 초기

void Game::gameStart()

{

setMaps();

setPlayer();

this->finished.assign(numOfStages, false);

this->currentStage = 0;

this->stepCount = 0;

this->pushCount = 0;

}

void Game::move(const int direction)

{

/\*

생략

\*/

// 목적지에 모두 상자가 차 있는지 체크

if (checkSuccess())

nextStage();

}

}

void Game::walk(const int y, const int x, const int nextY, const int nextX)

{

/\*

생략

\*/

stepCount++;

}

void Game::push( const int y, const int x, const int nextY, const int nextX,

const int afterY, const int afterX)

{

/\*

생략

\*/

stepCount++;

pushCount++;

}

void Game::nextStage()

{

finished[currentStage] = true; // 현 스테이지 클리어

currentStage++;

stepCount = 0; // 스텝 카운트 초기화

pushCount = 0; // 푸시 카운트 초기화

}

// 현재 모든 goal Position에 모두 상자(2)가 있는지 체크 하여 boolean 반환

bool Game::checkSuccess() const

{

int y, x;

for (int i = 0; i < goalPositions[currentStage].size(); i++)

{

y = goalPositions[currentStage][i].first;

x = goalPositions[currentStage][i].second;

if (maps[currentStage][y][x] != 2)

return false;

}

return true;

}

// 모든 스테이지의 finished 값이 true 인지 체크

bool Game::checkAllSuccess() const

{

for (int i = 0; i < finished.size(); i++)

if (finished[i] == false)

return false;

return true;

}

**< main.cpp파일 >**

int main()

{

/\*

생략

\*/

Game game;

game.gameStart();

// 게임이 완전히 끝날 때까지 반복

while (!game.checkAllSuccess())

{

/\*

생략

\*/

//현재 발자국 수, 상자 민 횟수 그리기

mvprintw(13, 12, "steps: %d", game.getStepCount());

mvprintw(13, 23, "pushes: %d", game.getPushCount());

refresh();

/\*

생략

\*/

// 반복 전에 전 스테이지와 맵이 달라질 경우, 맵이 겹치는 경우를 대비해 화면 지우고 다시 그리기

clear();

}

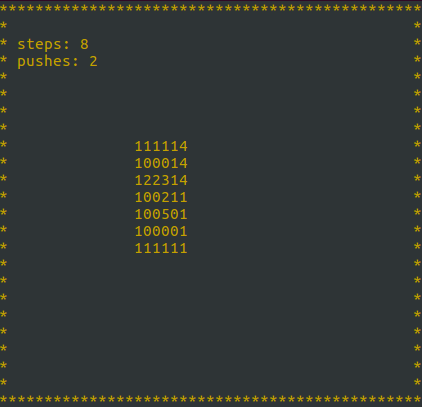
attroff(COLOR\_PAIR(1));

endwin();

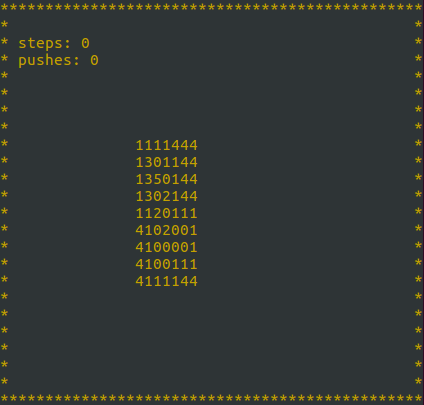
}

**< main.cpp파일 >**

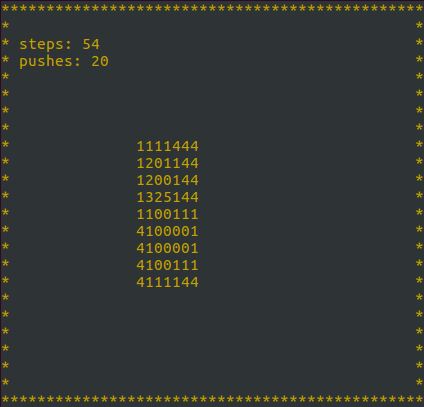
**1. 스텝 수, 푸시 수 표시. 스테이지1 마지막 푸시 전**



**2. 스테이지2 초기 화면**



**3. 스테이지2 마지막 푸시 전**



**[추가 기능]**

**1. 디자인**

**2. Q를 누르면 게임을 종료한다.**

**3. R을 누르면 해당 스테이지의 처음으로 돌아간다.**

**< Game.cpp 파일 >**

// 게임을 리셋하는 함수

void Game::reset()

{

this->playerPositions.clear();

setMaps();

setPlayer();

this->stepCount = 0;

this->pushCount = 0;

}

**< main.cpp파일 >**

int main()

{

/\*

생략

\*/

Game game;

game.gameStart();

//게임이 완전히 끝날때까지 반복

while (!game.checkAllSuccess())

{

/\*

생략

\*/

//입력받아 움직이기

int input = getch();

// Q를 입력 받을 시 종료

if (input == 'q')

break;

// R를 입력 받을 시 리셋

else if(input == 'r')

game.reset();

else

game.move(input);

//화면 지우기

clear();

}

attroff(COLOR\_PAIR(1));

endwin();

}

**< 최종 실행 화면 >**

