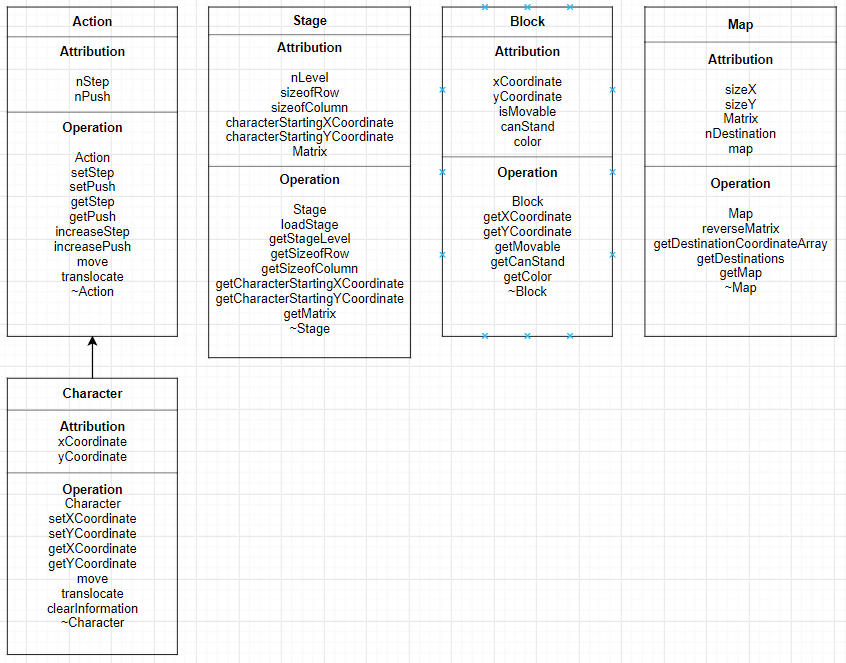
**C++ Project**

20181627 성주원, 20181631 송민준

**Curses 외부라이브러리 설명**

Curses 라이브러리는 두 종류가 있는데 ncurses와 pdcurses가 있다.  
우리 팀에서는 ncurses 대신 pdcurses를 이용했다.  
그 이유는 ncurses같은 경우 사용가능한 플랫폼이 한정돼 있는데 pdcurses같은 경우 크로스 플랫폼이 된다.  
ncurses 및 pdcurses의 성능 또는 기능차이 그리고 네이밍 차이는 전혀 없으며 단지 GNU 플랫폼에 한정되느냐 아니면 크로스 플랫폼이냐는 차이 밖에 없다.  
그래서 크로스 플랫폼을 지원하는 pdcurses를 사용했다.

**Class Diagram**



**<Action>**  
정해진 맵에서 다른 Class가 행동하기 위해 필수적인 요인들을 가지고 있는 Class

**Attribution**nStep: 캐릭터가 몇 번 걸었는지  
nPush: 캐릭터가 몇 번 나무 박스를 밀었는지

**Operation**  
increaseStep: nStep 1 증가  
increasePush: nPush 1 증가  
move: 특정 x, y 좌표로 이동  
translocate: 특정 블록 이동시키기

**<Character>**Action을 상속받는 Class

**Attribution**xCoordinate: 캐릭터의 x 좌표  
yCoordinate: 캐릭터의 y 좌표

**Operation**  
clearInformation: 캐릭터의 nPush, nStep 정보를 0으로 초기화 함

**<Stage>**Int Matrix로 입력받은 맵 정보 Class

**Attribution**nLevel: 맵 난이도  
sizeofRow: 배열의 row 크기  
sizeofColumn: 배열의 column 크기  
characterStartingXCoordinate: 캐릭터 시작 x 좌표  
characterStartingYCoordinate: 캐릭터 시작 y 좌표  
Matrix: 2차원 int 배열  
**Operation**loadStage: 1 ~ 4까지 난이도에 맞는 맵을 불러옴

**<Block>**게임에 표기될 블록 Class

**Attribution**xCoordinate: 블록의 x 좌표  
yCoordinate: 블록의 y 좌표  
isMovable: 움직이게 할 수 있는 블록인지  
canStand: 이 블록 위에 서 있을 수 있는지  
color: 블록의 색상

**<Map>**GUI 출력 및 Character의 행동을 확장성있도록 만들기 위한 Class

**Attribution**sizeX: x축 맵 크기  
sizeY: y축 맵 크기  
Matrix: Stage에 해당하는 Matrix(2차원 int 배열)  
nDestination: 해당하는 맵에 목적지 개수  
map: 2차원 Block 배열

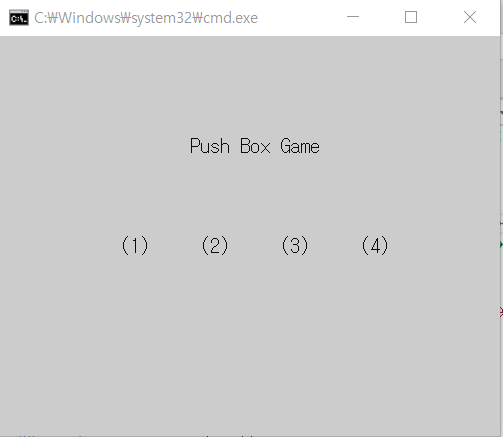
**Operation**  
reverseMatrix: Matrix를 역행렬로 만듦 (역행렬로 만드는 이유는 배열이 row부터 column으로 시작하기 때문임, 이럴 경우 캐릭터가 움직일 때 항상 x, y 좌표를 반대로 써줘야 하는데 역행렬을 만듦으로써 캐릭터의 움직임 좌표를 반대로 할 필요가 없어짐)  
역행렬을 만든 후 Matrix 숫자 정보를 가지고 map을 초기화 시킴  
getMap: 2차원 Block 클래스 배열을 가져옴

**Entire Flow**

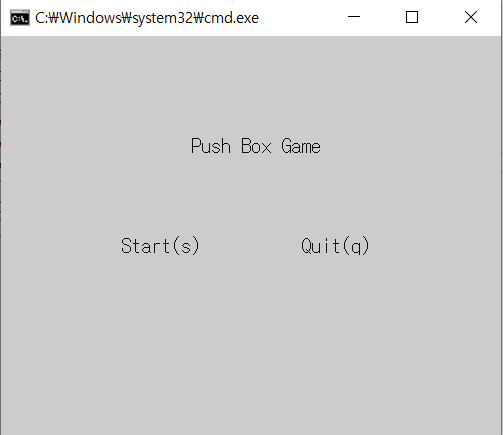
Stage 클래스에 각 난이도별 2차원 int 배열인 맵을 미리 초기화해 뒀으며 Stage 객체를 생성해서 난이도에 맞는 2차원 int 배열 맵으로 Stage attribution인 Matrix를 초기화 시킨다.  
초기화 시킨 이후 Map 클래스 생성자 parameter에 Stage가 가지고 있는 attribution들을 Getter 메서드를 이용해 가져와 Map 클래스를 생성한다.  
생성된 Map 클래스에서 getMap() 메서드를 이용해 2차원 Block 배열을 가져온다.  
Character는 2차원 Block 배열에서 이동시킨다.

**화면 구성**

**-시작화면**

****

**-스테이지 선텍화면**

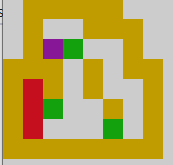
****

**-인게임 화면**

****

**단계별**

**1단계:** ncurses library 함수들을 사용하여 아래와 같은 2차원 배열로 표시된 push box map 을 game 화면으로 표시하는 프로그램을 완성한다. 그림의 세부사항은 각자 정한다. 아래 보이는 예의 배열에서 1은 벽을, 2는 상자를, 3은 목적지를 , 4는 바깥부분을 나타내므로 각각 다른 색으로 화면에 구분이 되게 표시한다.

****

**벽(노란색):** init\_pair(1, COLOR\_YELLOW, COLOR\_YELLOW);

**상자(초록색)**: init\_pair(2, COLOR\_GREEN, COLOR\_GREEN);

**목적지(빨간색)**: init\_pair(3, COLOR\_RED, COLOR\_RED);

**빈 공간(흰색)**: init\_pair(4, COLOR\_BLACK, COLOR\_WHITE);

**플레이어(보라색)**: init\_pair(5, COLOR\_MAGENTA, COLOR\_MAGENTA);

**init\_pair() 메서드로 앞으로 블록을 gui에 띄우는데 사용할 COLOR 조합을 각 숫자로 저장한다.**

**이 5가지의 종류를 제외한 나머지 공간은 bkgd() 메서드로 흰색 배경을 채운다**

bkgd(COLOR\_PAIR(4));

**Block 형태로 주어질 map 각각 블록들의 색깔에 따라 GUI를 구성한다.**

**이중 for문을 통해 모든 map의 원소들의 COLOR를 가져온 후 표시한다.**

void renderObject(Stage stage, Block\*\* map, Character user) {

clear();

for (int i = 0; i < stage.getStageSizeofColumn(); i++) {

int x = i \* 2;

for (int j = 0; j < stage.getStageSizeofRow(); j++) {

if ((i == user.getXCoordinate()) && (j == user.getYCoordinate())) {

//캐릭터 위치 표시

attron(COLOR\_PAIR(5));

mvprintw(j, x, " ");

mvprintw(j, x + 1, " ");

attroff(COLOR\_PAIR(5));

continue;

}

int color = map[i][j].getColor();

attron(COLOR\_PAIR(color));

mvprintw(j, x, " ");

mvprintw(j, x + 1, " ");

attroff(COLOR\_PAIR(color));

}

}

}

void renderObject(Stage stage, Block\*\* map, Character user): **인자 값으로 Stage , 이차원 Block, user 값이 입력된다.**

**/자료형은 추후 설명/**

**stage.getStageSizeofColumn() : 스테이지의 행 길이를 리턴한다.**

**stage.getStageSizeofRow() : 스테이지의 열 길이를 리턴한다.**

**Int color = map[i][j].getColor() : x = I, y = j 을 가진 원소의 COLOR 값을 가져온다. (여기서 map은 역행렬)**

**attron(COLOR\_PAIR(color)) : attroff(COLOR\_PAIR(COLOR)) 까지 COLOR\_PAIR(COLOR) 의 색조합으로 출력된다.**

**attroff(COLOR\_PAIR(color)**

**ex)**

attron(COLOR\_PAIR(5));

mvprintw(j, x, " ");

mvprintw(j, x + 1, " ");

attroff(COLOR\_PAIR(5));

**:COLOR\_PAIR(5)는 init\_pair(5, COLOR\_MAGENTA, COLOR\_MAGENTA) 에서 글자 색깔과 배경을 COLOR\_MAGENTA로 정해줬기 때문에 attroff(COLOR\_PAIR(5)) 일 때까지 보라색을 출력한다.**

**mvprintw(j,x, “ “) : 윈도우에서 y 는 j , x는 x에 해당하는 곳으로 이동 후 “ “를 출력한다. (정사각형에 가깝게 만들어 주기 위해 mvprintw(j,x+1,” “) 을 추가한다.)**

**//캐릭터 위치 표시**

if ((i == user.getXCoordinate()) && (j == user.getYCoordinate())) {

attron(COLOR\_PAIR(5));

mvprintw(j, x, " ");

mvprintw(j, x + 1, " ");

attroff(COLOR\_PAIR(5));

continue;

}

**i가 플레이어의 X 위치에 , j 가 플레이어의 Y위치에 도달한다면**

**COLOR\_PAIR(5) == magenta 색깔로 표시한다.**

**2단계:** 1단계의 맵 위에 캐릭터를 표시하고 화살표를 입력받아 캐릭터가 움직이도록 프로그램을 완성한다. 프로그램은 캐릭터의 이동방향에 벽이 있거나 상자가 두개 이상 연속으로 있으면 캐릭터가 움직이지 않도록 해야하고, 이동 방향에 상자가 한 개 있으면서 그 상자 반대편이 또다른 상자나 벽으로 막혀있지 않으면 상자도 캐릭터와 함께 이동 방향으로 한 칸 이동해야 한다.

**플레이어가 무엇을 누를지에 따라서 위 아래 왼쪽 오른쪽 방향 이동이 결정되도록 switch문을 사용한다.**

while (control != 'q') {

control = getch();

switch (control) {

case 'w':

destinationX = user.getXCoordinate();

destinationY = user.getYCoordinate() - 1;

if (stage.getMatrix()[destinationY][destinationX] == 3) {

user.move(destinationX, destinationY, map[destinationX][destinationY], map, 3);

}

else {

user.move(destinationX, destinationY, map[destinationX][destinationY], map, 0);

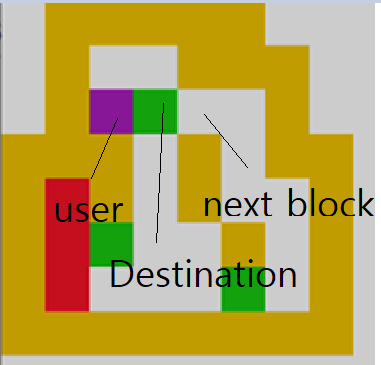
}

break;

**control = getch() : 자료형이 char인 control에 getch() 메서드로 키보드 입력을 받는다.**

**destinationX = user.getXCoordinate(); : DastinationX 에 현재 플레이어 x위치를 초기화 시킨다. (user.getXCoordinate() -1; 이면 왼쪽이동) user.getXCoordinate()는 user 클래스의 X값을 반환**

**destinationY = user.getYCoordinate();**

****

**\*상자 move**

if (stage.getMatrix()[destinationY][destinationX] == 3) {

user.move(destinationX, destinationY, map[destinationX][destinationY], map, 3);

**:이동할 칸이 상자일 경우 user.move(destinationX , destinationY , map[destinationX][destinationY], map,3) 호출.**

void Character::move(int destinationX, int destinationY, Block &block, Block \*\*map, int willStepBlockColor)

{

if (block.getCanStand()) {

this->xCoordinate = block.getXCoordinate();

this->yCoordinate = block.getYCoordinate();

this->increaseStep();

return;

}

if (block.getMovable()) {

translocate(destinationX, destinationY, block, map, willStepBlockColor);

}

}

**상자는 movable 하니까 translocate 호출**

void Character::translocate(int destinationX, int destinationY, Block &block, Block \*\*map, int willStepBlockColor) {

int differenceByOneBlockOnAxisToX = destinationX - this->xCoordinate;

int differenceByOneBlockOnAxisToY = destinationY - this->yCoordinate;

Block nextBlock = map[destinationX + differenceByOneBlockOnAxisToX][destinationY + differenceByOneBlockOnAxisToY];

if (nextBlock.getCanStand()) {

Block characterStoodBlock = map[this->xCoordinate][this->yCoordinate];

map[nextBlock.getXCoordinate()][nextBlock.getYCoordinate()] = block;

map[destinationX][destinationY] = Block(destinationX, destinationY, false, true, willStepBlockColor);

move(destinationX, destinationY, map[destinationX][destinationY], map, willStepBlockColor);

this->increasePush();

}

else {

return;

}

}

**int differenceByOneBlockOnAxisToX = destinationX - this->xCoordinate : 목적지와 현재 x 위치 의 차를 저장**

**Block nextBlock = map[destinationX + differenceByOneBlockOnAxisToX][destinationY + differenceByOneBlockOnAxisToY]**

**: 현재 방향으로 목적지에서 다음 칸으로 갔을 때의 Block을 nextBlock에 초기화**

if (nextBlock.getCanStand()) {

Block characterStoodBlock = map[this->xCoordinate][this->yCoordinate];

map[nextBlock.getXCoordinate()][nextBlock.getYCoordinate()] = block;

map[destinationX][destinationY] = Block(destinationX, destinationY, false, true, willStepBlockColor);

move(destinationX, destinationY, map[destinationX][destinationY], map, willStepBlockColor);

this->increasePush();

}

else {

return;

}

**만약 다음 블록이 상자 혹은 벽일 경우 유지,**

**다음 블록이 빈 공간 혹은 목적지일 경우 캐릭터와 상자가 한칸 이동**

**실행화면**

****

****

****

**3단계:** 2단계 프로그램에서 step 횟수 (캐릭터가 이동한 횟수) 와 push 횟수 (상자가 움직인 횟수) 를 화면에 보여주고 모든 상자가 목적지에 도달하면 게임을 끝내고 다음 map 으로 넘어가서 다시 게임을 시작하는 프로그램을 완성하라.

**-Step , Push , Stage 를 표시할 window 생성**

refresh();

WINDOW \* state = newwin(8, 15, 0, 30);

mvwprintw(state, 0, 0, "State Bar");

mvwprintw(state, 2, 0, "Stage: ");

buff[256];

sprintf\_s(buff, "%d", stage.getStageLevel());

mvwprintw(state, 2, 8, buff);

mvwprintw(state, 4, 0, "push: ");

char buff2[256];

sprintf\_s(buff2, "%d", user.getPush());

mvwprintw(state, 4, 7, buff2);

mvwprintw(state, 6, 0, "step: ");

char buff3[256];

sprintf\_s(buff3, "%d", user.getStep());

mvwprintw(state, 6, 7, buff3);

wrefresh(state);

**Window \* state = newwin(8,15, 0 ,30): 새로운 윈도우를 생성해준다.**

**newwin의 인자값은 (윈도우의 최대 Y 값, 윈도우의 최대 X 값, 윈도우 시작 Y위치 , 윈도우 시작 X위치)**

**mvwprintw(state, 0, 0, "State Bar"); : state 윈도우 에서 Y = 0 , X = 0 의 위치에서 “state bar” 생성**

**stage.getStageLevel() : 스테이지의 인덱스를 리턴한다.**

**//In stage class**

int Stage::getStageLevel() const {

return this->nLevel;

}

**user.getPush() : user의 Push 횟수를 리턴한다.**

**//in Action class**

int Action::getPush() const

{

return this->nPush;

}

**user.getStep(): user의 Step 횟수를 리턴한다.**

**//in Action class**

int Action::getStep() const

{

return this->nStep;

}

**wrefresh(state) : refresh()와 비슷한 개념으로 창에서 보이게끔 하기 위해 인자 값의 윈도우를 refresh 시킨다.**

**실행 화면**

****

****

****

**\*게임이 끝나는 조건은 모든 목적지가 모두 나무 상자여야 하는 것.**

**그리고 게임이 끝나면 다음 스테이지가 아닌 스테이지 선택 창이 나오도록**

int\*\* red = mapCreator.getDestinationCoordinateArray();

**~**

IsAllDestinationWoodBox = true;

for (int i = 0; i < mapCreator.getDestinations(); i++) {

if (map[red[i][0]][red[i][1]].getColor() != 2) {

isAllDestinationWoodBox = false;

break

}

}

if (isAllDestinationWoodBox) {

user.clearInformation();

clear();

break;

}

**int\*\* red = mapCreator.getDestinationCoordinateArray(); : 맵의 목적지가 있는 부분을 x 와 y 값으로 저장한 2차원 정수형 배열**

**isAllDestinationWoodBox = true; : 나무 상자가 모두 목적지에 있는지 여부를 확인해주는 boolean값**

**if(map[red[i][0]][red[i][1]].getColor() !=2) : 목적지의 현재 상태가 나무상자가 아니면 isAllDestinationWoodBox를 false 후 break; 즉 다음 입력을 기다린다. 만약 for문 이후 true라면 게임이 끝나고**

if ((isAllDestinationWoodBox == true) || (control == 'q')) {

clear();

mvprintw(5, windowX / 3 + 3, "Push Box Game");

mvprintw(10, windowX / 4, "(1)");

mvprintw(10, windowX / 4 + 8, "(2)");

mvprintw(10, windowX / 4 + 16, "(3)");

mvprintw(10, windowX / 4 + 24, "(4)");

control = getch();

}

**으로 스테이지 선택 화면 재구성 시킨다.**

**실행화면**

