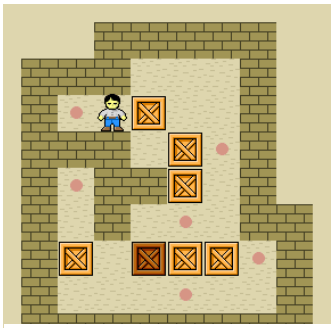
Push Box Game 프로젝트

20181653 이강희

20181656 이민종

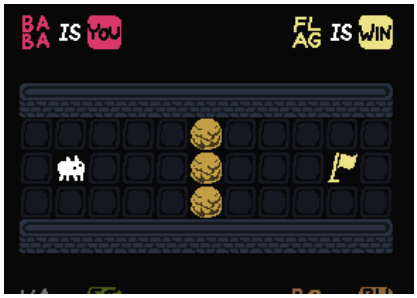
I Push Box Game

- 모든 박스들을 목표 지점으로 옮기는 것이 목표인 퍼즐 게임이다
- 1981년 일본에서 개발된 Sokoban이라는 게임이 시초.
- 컴퓨터, 스마트폰, 콘솔 게임기 등에 다양한 버전의 Push Box Game이 있다.
- 사각형 타일이 아닌 정육각형 타일을 사용하거나, 한번에 여러 개의 상자를 밀 수 있거나, 벽을 파괴하는 등의 다양한 변형 Sokoban 게임들이 있다.



Sokoban

Hexoban

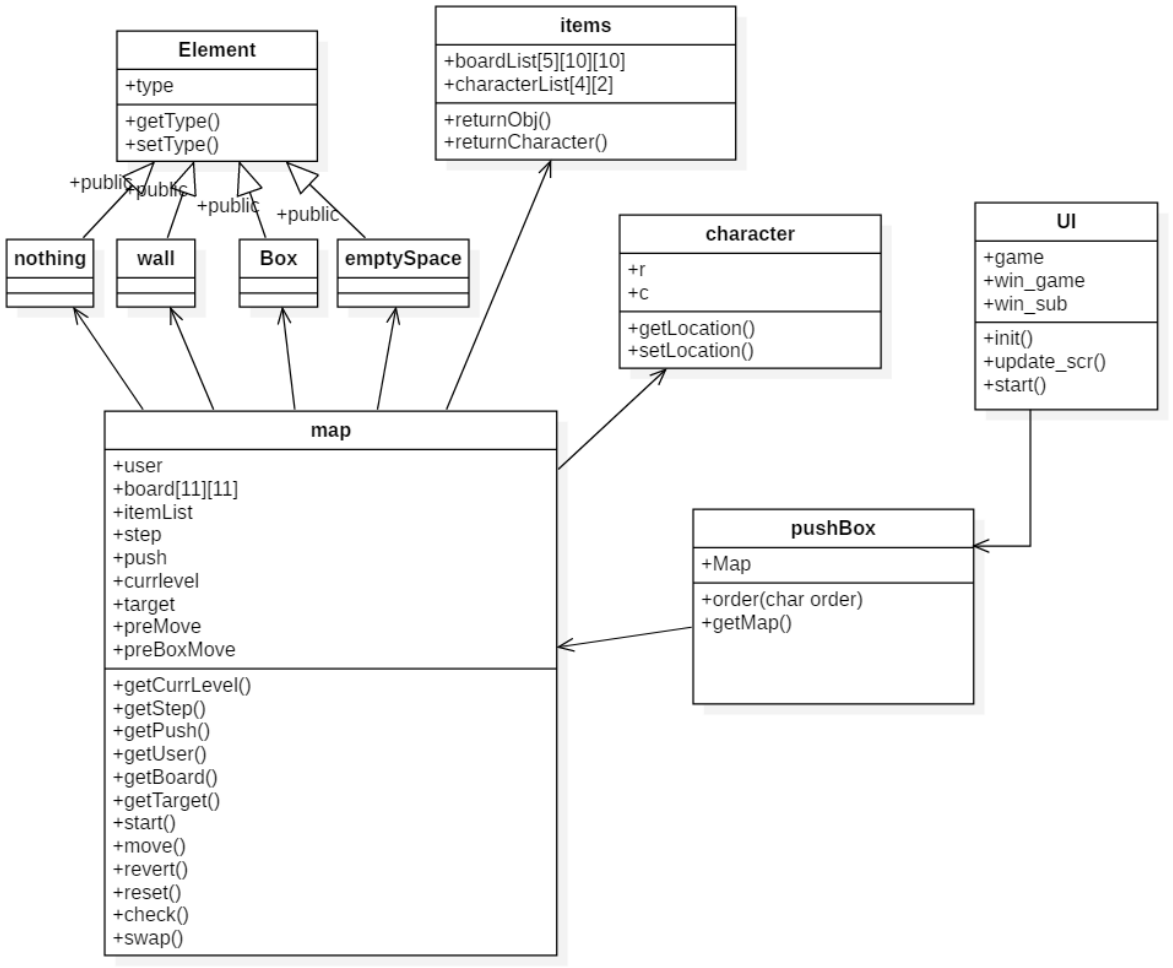


Baba is you

게임 규칙

- 게임은 사각형 보드에서 진행된다.
- 각 칸들은 빈 공간, 박스, 목표 지점, 플레이어 중 한가지로 구성되어 있다.
- 플레이어는 상하좌우의 빈칸으로 이동할 수 있다.
- 만약 플레이어가 박스가 있는 칸으로 이동하면, 박스를 밀 수 있다. 만약 박스 다음 칸에 벽이나 박스가 있으면 밀 수 없다.
- 퍼즐은 모든 박스가 목표 지점으로 이동하면 풀린다.
- 이 외에도 이번 과제에서 Step, Push 횟수를 세는 기능, 맵을 Reset 하는 기능, 플레이어 행동을 되돌리는 기능을 추가하였다.

Ⅱ Logic 구성



기본적인 로직의 구성은 위의 UML과 동일하다. 각 좌표의 객체들을 하나의 오브젝트로 표현하기 위해 Element라는 부모 클래스를 만들었고, 그 속성인 type을 초기화하는 자녀 클래스 nothing, wall, Box, emptySpace 를 만들었다. 또한 캐릭터가 어디있는지 알 수 있도록 character라는 클래스를 하나 만들었다. 이 모든 클래스를 가지고 map이 게임의 전체적인 진행을 관리한다.

클래스 map은 보드판에 대한 데이터를 가지고 있는 클래스 items를 통해 아이템을 가져와 그에 맞게 element subclass를 board에 배치하고, 이 board를 가지고 게임을 진행하게 된다.

pushBox는 사용자의 입력과 출력, 게임의 로직을 연결하는 controller 역할을 하는 클래스다.

UI는 사용자의 입력과 그에 맞는 출력을 수행하는 클래스다.

정리하면, 게임이 시작되면 pushBox에서 map에게 초기화를 명령하고 그에 맞게 UI에서 출력한다. 이후 사용자의 입력이 들어오면 UI, pushBox를 거쳐 map에서 그에 맞는 동작을 수행한 뒤, 그 내용을 UI에서 출력한다.

class 세부 내용 설명

1. Element 클래스 (subClass 포함)

- **type** : 그 객체의 타입을 나타내는 변수.
- **getType()** : type을 반환하는 함수
- **setType()** : type을 설정하는 함수

```
#include "elements.h"


int items::returnObj(int level, int i, int j) const {
        return boardList[level][i][j];
};

int* items::returnCharacter(int level) {
        if (level >= 5) return NULL;
        return characterList[level];
};

char element::getType() const { return type; }
void element::setType(char t) { type = t; }

nothing::nothing() {
        setType('n');
}

wall::wall() {
        setType('w');
}
```

element.cpp

```
Box::Box() {
        setType('b');
}
emptySpace::emptySpace() {
        setType('e');
}

void character::getLocation(int l[]) {
        l[0] = r;
        l[1] = c;
}
void character::setLocation(int row, int col) {
        r = row;
        c = col;
}
```

element.cpp 뒷부분

2. items 클래스

- **boardList** : 기본 보드판에 대한 내용을 담고 있다. 총 5개의 보드판으로 0은 빈공간, 1은 벽, 2는 상자, 3은 상자를 놓아야 할 공간, 4는 게임 밖 공간을 의미한다.
- **character** : 각 보드판에 대한 캐릭터의 위치를 담고 있다.
- **returnObj()** : 보드판의 원하는 좌표의 객체에 대해 반환하는 함수
- **returnCharacter()** : 원하는 보드판의 캐릭터 위치를 반환하는 함수

3. character 클래스

- **r** : 행의 위치를 담고 있다.
- **c** : 열의 위치를 담고 있다.
- **getLocation()** : 현재 위치를 가져오는 함수
- **setLocation()** : 현재 위치를 수정하는 함수

```
class character {
        int r, c;

public:
        void getLocation(int l[]);
        void setLocation(int row, int col);
};
```

character.h

```
void character::getLocation(int l[]) {
        l[0] = r;
        l[1] = c;
}
void character::setLocation(int row, int col) {
        r = row;
        c = col;
}
```

character.cpp

4. map 클래스

멤버 변수

- **user** : character 클래스의 객체
- **board** : element의 리스트로 게임판에 대한 정보를 담고 있다.
- **itemList** : items 클래스의 객체
- **step** : 현재 움직인 횟수
- **push** : 현재 박스를 민 횟수
- **currlevel** : 현재 맵에 대한 레벨
- **target** : 박스를 놓아야 할 공간에 대한 정보를 담고 있다.
- **preMove** : 이전에 어떻게 움직였는지에 대한 정보를 담고 있는 stack 객체
- **preBoxMove** : 이전에 박스가 언제 움직였는지에 대한 정보를 담고 있는 stack 객체

```
class map {
        character user;
        element board[11][11];
        items itemList;
        int step, push, currlevel;
        int target[13];
        /*char preMove;
        bool preBoxMove;*/
        stack<char> preMove;
        stack<bool> preBoxMove;

public:
        map();

        int getCurrLevel() const;
        int getStep() const;
        int getPush() const;
        character getUser();
        element(*getBoard())[11];
        void getTarget(int *ptr);

        void start(int level = 0);
        void swap(element &e1, element &e2);
        bool move(char order);
        void reset();
        bool revert();
        bool check();
};
```

map.h

메서드

- **getCurrLevel()** : 현재 level에 대해 반환
- **getStep()** : 현재 이동한 횟수를 반환
- **getPush()** : 현재 박스를 민 횟수를 반환
- **getUser()** : character(user)의 위치를 반환
- **getTarget()** : 박스를 놓아야할 공간을 반환

```
int map::getCurrLevel() const { return currlevel; }
int map::getStep() const { return step; }
int map::getPush() const { return push; }
character map::getUser() { return user; }
element(*map::getBoard())[11]{ return board; }
void map::getTarget(int *ptr) {
        int n = target[0];
        ptr[0] = target[0];
        for (int i = 1; i < 2 * n + 1; i += 2) {
                ptr[i] = target[i];
                ptr[i + 1] = target[i + 1];
        }
}
```

map.cpp

메인 로직과 관련된 메서드

- **start()**

  인자로 넣은 level에 맞은 보드판을 가져오고 모든 변수를 초기화시킨다.

  level과 preMove, preBoxMove, step, push, user를 간단히 초기화하며, 2차원 배열인 board를 for문을 사용해 각 자리에 맞는 element 객체를 넣는 모습이다.

```
void map::start(int level) {
        currlevel = level;
        target[0] = 0; int n = 1;
        for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                        int temp = itemList.returnObj(level, i, j);
                        if (temp == 0) board[i][j] = emptySpace();
                        else if (temp == 1) board[i][j] = wall();
                        else if (temp == 2) board[i][j] = Box();
                        else if (temp == 3) {
                                board[i][j] = emptySpace();
                                target[0]++;
                                target[n] = i;
                                target[n + 1] = j;
                                n += 2;
                        }
                        else board[i][j] = nothing();
                }
        }
        int* rc = itemList.returnCharacter(currlevel);
        user.setLocation(rc[0], rc[1]);
        step = 0;
        push = 0;
        while(!preMove.empty()) preMove.pop();
        while(!preBoxMove.empty()) preBoxMove.pop();
}
```

map.cpp / start()

- **move()** : 캐릭터가 움직이는 것을 구현한 함수

  인자로 넣은 방향에 맞는 이동을 수행한다. 아래 왼쪽 그림에서 볼 수 있듯이 인자로 order를 받아 이동한다. 아래의 w, 위로가는 것을 예시로 들자면, 만약 위가 비어있다면 그 방향으로 이동하게 된다. 하지만 만약 박스가 있다면, 그 박스 다음 자리가 비어 있으면 이동하며, 아니라면 이동하지 않게 된다.

  이와 같은 코드를 각 방향, w, a, s, d에 대해 이동하며, 이동과 동시에 push, move를 업데이트 한다. 이동한다면 true를, 아니면 false를 리턴한다.

```
bool map::move(char order) {
	int l[2];
	user.getLocation(l);
	int r = l[0], c = l[1];
	if (order == 'w' && r != 0) {
		if (board[r - 1][c].getType() == 'e') preBoxMove.push(false);
		else if (board[r - 1][c].getType() == 'b' && r != 1) {
			if (board[r - 2][c].getType() == 'e') {
				swap(board[r - 2][c], board[r - 1][c]);
				/*preBoxMove = true;*/
				preBoxMove.push(true);
				push++;
			}
			else return false;
		}
		else return false;
		user.setLocation(r - 1, c);
	}
```

map.cpp / move 함수 코드 앞쪽

```
		else return false;
		preMove.push(order);
		/*preMove = order;*/
		step++;
		return true;
}
```

map.cpp / move 함수 뒤쪽

- **reset()**

  진행되고 있는 게임을 초기화시키고, 현재 level의 처음으로 돌아간다.

  이를 구현하기 위해 reset함수는 처음 start()를 실행한 상황으로 되돌아가면 되는 것이기 때문에 간단하게 start(currlevel)을 통해 구현하게 되었다.

```
void map::reset() {
	start(currlevel);
}
```

map.cpp / reset

- **revert()**

  수행했던 바로 이전의 이동 전으로 돌아간다. (예로 한번 위로 올라갔다면 다시 아래로 내려오는 함수)

  아래의 코드를 보면,

  현재 user의 위치를 가져온 뒤, preMove와 preBoxMove 스택이 비어있다면 동작하지 않는다. 만약 비어있지 않다면, move 함수와 비슷한 방식으로 이전의 이동을 preMove에서 꺼내와 수행한다.

  예로 이전 이동이 w, 즉 위로 이동했다면 아래로 이동하는데, 만약 그때 박스를 밀었다면 그 박스도 같이 내려와야 하기 때문에 preBoxMove의 top 또한 같이 고려한다.

  이를 모든 방향(a, s, d) 모두 수행한다.

  이동했다면 true를, 안했다면 false를 리턴한다.

```
bool map::revert() {
        int l[2];
        user.getLocation(l);
        int r = l[0], c = l[1];
        if (preMove.empty() || preBoxMove.empty()) return false;
        if (preMove.top() == 'w') {
                if (preBoxMove.top() && r != 0) {
                        board[r][c] = Box();
                        board[r - 1][c] = emptySpace();
                        push--;
                }
                user.setLocation(r + 1, c);
        }
```

map.cpp / revert 메서드 코드 앞쪽

```
        else return false;
        step--;
        preMove.pop();
        preBoxMove.pop();
        return true;
}
```

map.cpp / revert 메서드 코드 뒤쪽

- **check()**

게임이 끝났는지 확인. 즉, 모든 상자가 놓아져야 할 곳에 놓여져 있는지 확인

```
bool map::check() {
        int * temp = target;
        int n = temp[0];
        for (int i = 1; i < n * 2 + 1; i += 2) {
                if (board[temp[i]][temp[i + 1]].getType() != 'b') return false;
        }
        return true;
}
```

map.cpp / check 함수

구현은 상자가 놓여야 할 위치를 담고 있는 target 객체를 받아와 모든 위치 위에 있는 element 객체의 type이 box라면 true를 리턴하며, 아니면 false 를 리턴한다.

- **swap()**

인자로 전달된 두개의 element 객체의 자리를 바꾸는 함수. 다른 swap 함수와 똑같이 구현되었다.

- **좀 더 자세한 내용은 첨부된 code를 참고해주세요**

5. pushBox 클래스

- Map : map객체
- order() : 유저의 명령을 map에 전달해주는 함수
- getMap() : map 객체를 반환하는 함수

```h
class pushBox {
private:
	map Map;

public:
	pushBox();

	void order(char order);
	map getMap();
	void getL(int l[]);
};
```

pushBox.h

```cpp
void pushBox::order(char order) {
	if (order == 'b') Map.revert();
	else if (order == 'r') Map.reset();
	else {
		if (Map.move(order)) {
			if (Map.check()) {
				if (Map.getCurrLevel() == 4) Map.start(0);
				Map.start(Map.getCurrLevel() + 1);
			}
		}
	}
}

map pushBox::getMap() {
	return Map;
}

void pushBox::getL(int l[]) {
	Map.getUser().getLocation(l);
}
```

pushBox.cpp

6. UI 클래스

사용자에게 Push Box Game 화면을 보여주고, 입력을 받는 기능을 수행하는 것을 UI 라고 하며, ncurses로 구현한다

3가지 윈도우로 구성되는데,

전체를 감싸는 **기본 윈도우**

게임 화면을 표시하는 **Game 윈도우**

점수를 표시하는 **Sub 윈도우**

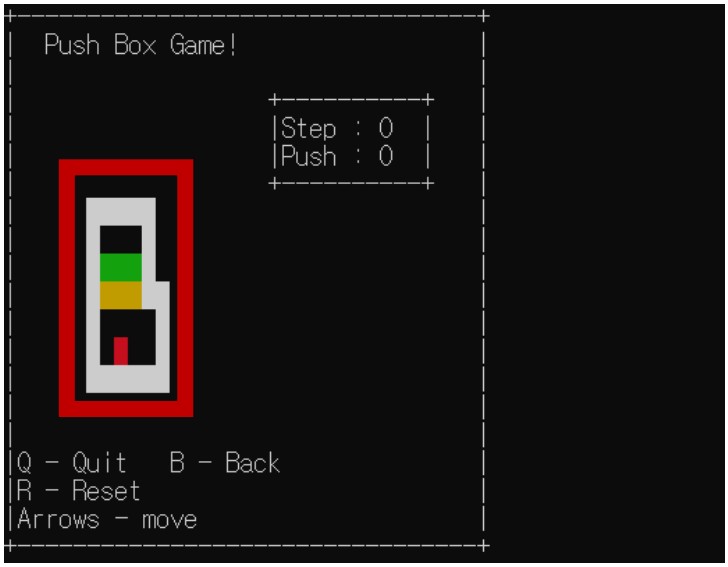
가 있다.

빨간색 – 캐릭터

노란색 – 상자

초록색 – 목표지점

흰색 – 벽



```h
#ifndef UI_H
#define UI_H
#include "pushBox.h"
#include <ncurses.h>
using namespace std;

enum palette { DEFAULT, CHARACTER, BOX = 'b', WALL = 'w', EMPTY = 'e', NOTHING = 'n', TARGET };

class UI {
public:
        UI(); // 생성자
        ~UI(); // 소멸자
        void init(); // ncurses 모드 시작
        void update_scr(); // 게임 로직을 가져와서 출력
        void start(pushBox *game); // 게임 로직을 가져와서 게임 시작

private:
        pushBox *game; // 게임 로직
        WINDOW *win_game; // ncurses game 윈도우
        WINDOW *win_sub; // ncurses sub 윈도우
};

#endif
```

UI.h

생성자 UI()

- init() 함수를 호출한다

```cpp
UI::UI() {
        init();
}
```

void init()

- ncurses 모드를 시작하고, 기본 화면을 출력한다.
- 사물들의 색 정의
- 윈도우 정의

```
13  void UI::init() {
14          // Initialize ncurses
15          // ncurses 모드 시작
16          initscr();
17          keypad(stdscr, TRUE); // 방향키 같은 특수키도 입력받을 수 있도록 keypad 설정
18          curs_set(0); // 화면에 보이는 커서 사라짐
19          noecho(); // 키보드 입력을 출력하지 않음
20
21          // Initialize colors
22          // DEFAULT, CHARACTER, ... 등등은 UI.h에서 선언한 enum 자료형, 즉 int 자료형.
23          start_color();
24          init_pair(DEFAULT, COLOR_WHITE, COLOR_BLACK);
25          init_pair(CHARACTER, COLOR_RED, COLOR_RED);
26          init_pair(BOX, COLOR_YELLOW, COLOR_YELLOW);
27          init_pair(WALL, COLOR_WHITE, COLOR_WHITE);
28          init_pair(EMPTY, COLOR_BLACK, COLOR_BLACK);
29          init_pair(NOTHING, COLOR_BLACK, COLOR_BLACK);
30          init_pair(TARGET, COLOR_GREEN, COLOR_GREEN);
31
32          // Initialize windows
33          // 윈도우들의 크기와 위치를 설정하고, border를 설정한다.
34          resize_term(20, 35);
35          border('|', '|', '-', '-', '+', '+', '+', '+');
36          win_game = newwin(12, 12, 3, 3);
37          wborder(win_game, ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ');
38          win_sub = newwin(4, 12, 3, 19);
39          wborder(win_sub, '|', '|', '-', '-', '+', '+', '+', '+');
40
41          // Text
42          // 기본 텍스트 출력
43          mvprintw(1, 3, "Push Box Game!");
44          mvprintw(16, 1, "Q - Quit   B - Back");
45          mvprintw(17, 1, "R - Reset");
46          mvprintw(18, 1, "Arrows - move");
47          mvwprintw(win_game, 1, 1, "win_game");
48          refresh();
49          wrefresh(win_game);
50          wrefresh(win_sub);
51  }
```

UI.cpp / init()

void start(pushBox *game)

- 게임 로직을 입력 받아서 게임을 시작함
- 사용자의 입력을 받아서 게임 로직에 전달하고, 화면을 업데이트 하는 과정을 무한 반복함.
- 사용자가 q를 입력하면 게임 종료

```
83  void UI::start(pushBox *game) {
84          this->game = game;
85          while (true) {
86                  update_scr(); // 화면을 업데이트함
87                  int key = getch(); // 사용자 키 입력을 기다림
88                  if (isupper(key)) key = tolower(key); // 알파벳 대문자를 소문자로 변경
89                  if (258 <= key && key <= 261) { key = "swad"[key - 258]; } // 방향키를 wasd로 변경
90                  if (key == 'q') break; // Q를 입력하면 게임 종료
91                  game->order(key); // 게임 로직에 사용자 입력 전달
92          }
93          this->game = NULL;
94  }
```

UI.cpp

void update_scr()

- 게임 화면을 업데이트 하는 메소드.
- 게임 로직으로부터 맵, 캐릭터 위치, Step, Push 횟수 등을 가져와서 출력한다.

```
53  void UI::update_scr() {
54          element(*board)[11] = game->getMap().getBoard(); // 게임 로직으로부터 맵을 가져옴
55          // 2차원 배열을 출력함
56          for (int i = 0; i < 10; ++i) {
57                  for (int j = 0; j < 10; ++j) {
58                          char type = board[i][j].getType();
59                          wattron(win_game, COLOR_PAIR(type));
60                          mvwprintw(win_game, i + 1, j + 1, "%c", type);
61                  }
62          }
63          // 목표지점을 출력함
64          int target[13];
65          game->getMap().getTarget(target);
66          wattron(win_game, COLOR_PAIR(TARGET));
67          for (int i = 0; i < target[0]; ++i) {
68                  int r = target[i * 2 + 1];
69                  int c = target[i * 2 + 2];
70                  if (board[r][c].getType() == 'b') continue;
71                  mvwprintw(win_game, r + 1, c + 1, "T");
72          }
73
74          // 플레이어 위치를 표시함
75          int player[2];
76          game->getL(player);
77          wattron(win_game, COLOR_PAIR(CHARACTER));
78          mvwprintw(win_game, player[0] + 1, player[1] + 1, "C");
79          wrefresh(win_game);
80
81          // Step, Push 횟수를 출력함
82          mvwprintw(win_sub, 1, 1, "Step : %-3d", game->getMap().getStep());
83          mvwprintw(win_sub, 2, 1, "Push : %-3d", game->getMap().getPush());
84          wrefresh(win_sub);
85  }
```

UI.cpp

소멸자 ~UI()

- 동적 할당된 윈도우들을 삭제시킴
- ncurses 모드를 끝냄
- 표준 출력(cout) 으로 터미널에 게임이 끝남을 출력

```
7   UI::~UI() {
8           delwin(win_game);
9           delwin(win_sub);
10          endwin();
11          cout << "Game Finished\n";
12  }
```

7. make file

리눅스에서 make 명령을 통해 자동적으로 컴파일을 실행하기 위해 make file을 만들었다.

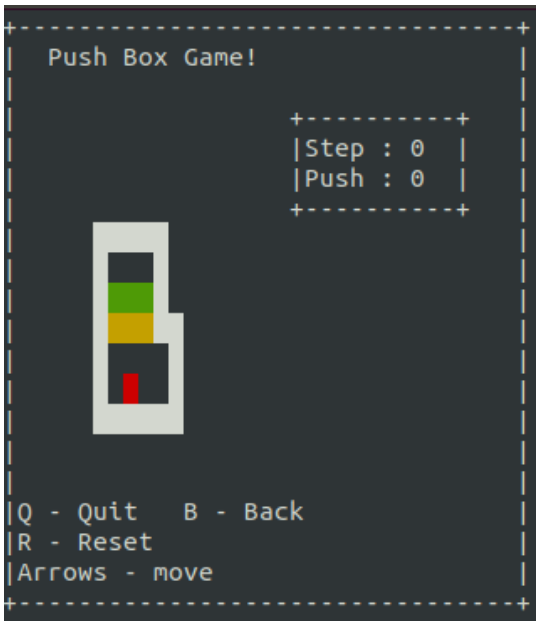
아래의 코드처럼 구현하였고, 터미널에서 make를 명령하면 위의 순서대로 컴파일하게 된다.

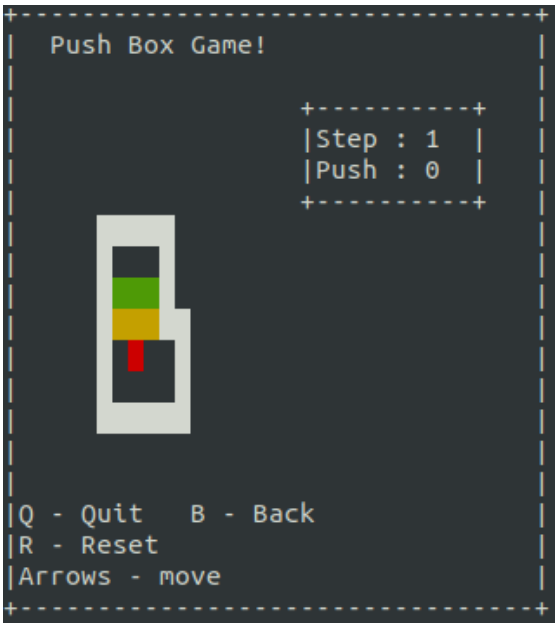
```
all:pushbox

pushbox:elements.o map.o pushBox.o UI.o main.o
	g++ -o pushbox elements.o map.o pushBox.o UI.o main.o -lncurses

elements.o:elements.cpp
	g++ -c -o elements.o elements.cpp

map.o:map.cpp
	g++ -c -o map.o map.cpp

pushBox.o:pushBox.cpp
	g++ -c -o pushBox.o pushBox.cpp

UI.o:UI.cpp
	g++ -c -o UI.o UI.cpp

main.o:main.cpp
	g++ -c -o main.o main.cpp

clean:
	rm -f *.o
```

아래는 실제 make 하는 모습이다.

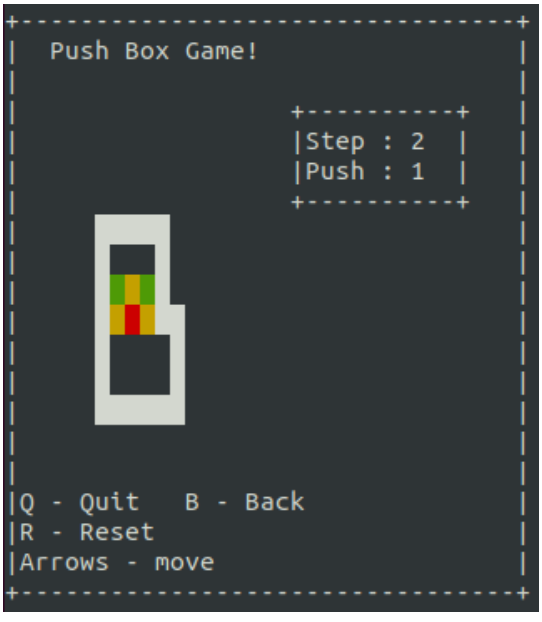
```
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$ ls
Makefile  UI.h          elements.h  main.cpp  map.h        pushBox.h
UI.cpp    elements.cpp  main        map.cpp   pushBox.cpp
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$ make
g++ -c -o elements.o elements.cpp
g++ -c -o map.o map.cpp
g++ -c -o pushBox.o pushBox.cpp
g++ -c -o UI.o UI.cpp
g++ -c -o main.o main.cpp
g++ -o pushbox elements.o map.o pushBox.o UI.o main.o -lncurses
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$ ls
Makefile  UI.o          elements.o  main.o   map.o        pushBox.o
UI.cpp    elements.cpp  main        map.cpp  pushBox.cpp  pushbox
UI.h      elements.h    main.cpp    map.h    pushBox.h
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$ make clean
rm -f *.o
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$ ls
Makefile  UI.h          elements.h  main.cpp  map.h        pushBox.h
UI.cpp    elements.cpp  main        map.cpp   pushBox.cpp  pushbox
hugh@hugh-ThinkPad-T470:~/Push-Box-Game/make$
```

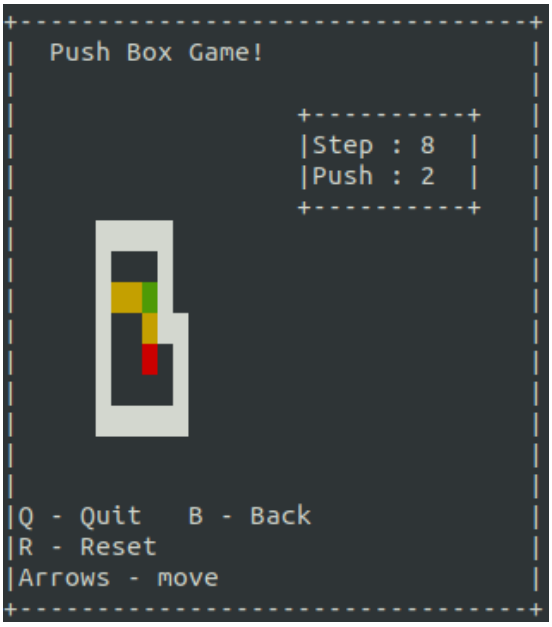
Ⅲ 실제 구현 결과





처음에 실행하면 이와 같이 뜬다. 빨간색이 사용자이며, 흰색은 벽, 노란색은 상자, 초록색은 상자를 놓아야할 위치다. 화살표를 누르면(혹은 WASD) 위 아래로 움직일 수 있다. 첫번째에서 2번째 사진으로는 윗방향 화살표를 눌려 간 모습이다. 그에 따라 step이 1 커졌다.

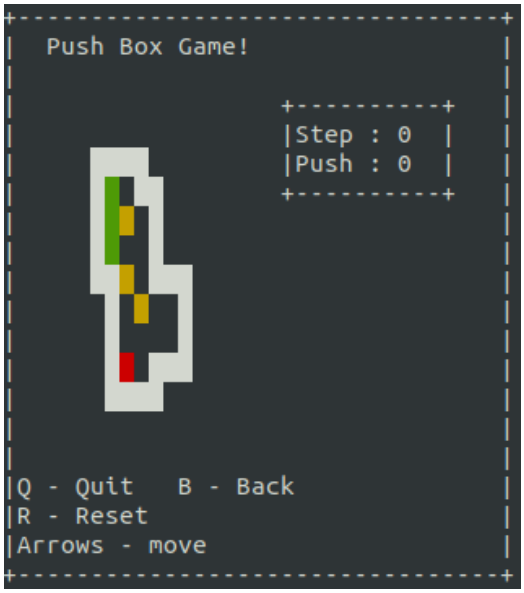




위에 상자가 있을 때, 윗버튼을 누르면 상자와 같이 위로 올라가게 된다. 초록색 부분은 상자가 있어야 할 부분, 즉 우리가 상자를 놓아야 하는 부분이다. 이에 따라 push가 1 커진 모습이다.

(여기서 b를 눌러 뒤로 가는 것을 명령한다면, 4 -> 3 -> 2 -> 1 순으로 되돌아 갈 것이며

4번째 그림에서 상자를 위로 올려 모든 상자를 놓아야 할 위치에 놓는다면 다음 map으로 이동하게 된다.



이렇게 다음 map으로 넘어온 모습이다.