

C++ 프로그래밍
Battleship 1단계 프로젝트 보고서

학번: 20171676

학과: 소프트웨어학부

이름: 이정우

1. 클래스 정의

게임판을 그리는 GameBoard 클래스, 전체적인 게임을 관리하는 GameManager 클래스, 각 배를 구현하는 Ship 클래스와 그것을 상속받는 Aircraft, Battleship, Cruiser, Destroyer 클래스, 그리고 각 플레이어인 Attacker, Defender 클래스로 이루어져 있다.

1-1. GameBoard.h 헤더 파일

```
// C++ BattleShip 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#pragma once
#include <ncurses.h>

// GameBoard 정의. 게임판 출력, 입력 확인 담당.
class GameBoard{
private:
    WINDOW *DEFENDER; // Defender 맵 윈도우
    WINDOW *ATTACKER; // Attacker 맵 윈도우
    WINDOW *STATUS; // Status 윈도우
    WINDOW *INPUT; // Input 윈도우
    WINDOW *RESULT; // Input Result 윈도우
    char position[2]; // Position input 저장 변수
    const int MAP_SIZE = 8; // 맵 크기
public:
    void init(); // 기초 세팅 초기화

    // 전체적인 판을 그리는 함수
    void render();
    void drawDefenderMap();
    void drawAttackerMap();
    void drawStatusWindow();
    void drawInputWindow();

    void refreshDefenderMap(int **[8]); // Defender 윈도우를 refresh하는 함수
    void refreshAttackerMap(int **[8]); // Attacker 윈도우를 refresh하는 함수
    bool inputAttack(int **[8], int **[8]); // 좌표 입력을 받아 Hit, Miss 여부를 확인, 잘못된 입력이면 false 리턴
    bool checkHit(int, int, int **[8]); // 좌표가 Hit이면 true, Miss이면 false 리턴
    void refreshInput(const char*); // Input 윈도우를 refresh하는 함수
    void refreshStatus(int, bool, bool, bool, bool, bool); // Status 윈도우를 refresh하는 함수
    void exitGame(); // 게임을 종료하는 함수
};
```

GameBoard 클래스는 게임판을 그리는 ncurses 를 이용하는 모든 부분을 담당한다.

변수와 메소드에 관한 설명은 주석을 보면 알 수 있다.

1-2. GameManager.h 헤더 파일

```
// C++ BattleShip 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#pragma once
#include "Defender.h"
#include "Attacker.h"
#include "GameBoard.h"

// GameManager 정의. 게임 플레이어, 게임 보드, 여러 정보들을 관리
class GameManager{
private:
    GameBoard m_gameboard; // GameBoard 생성
    Defender m_defender; // Defender 생성
    Attacker m_attacker; // Attacker 생성
    int turn; // 현재 턴을 저장할 변수
public:
    GameManager(); // 생성자
    void play(); // 게임 시작 함수
    bool allDestroy(); // 모든 배가 파괴되었는지 여부 확인하는 함수. 모두 파괴되었으면 true 리턴.
};
```

GameManager 클래스는 게임의 전체적인 진행과 상황을 관리하는 역할이다.

변수와 메소드에 관한 설명은 주석을 보면 알 수 있다.

1-3. Defender.h 헤더 파일

```
// C++ Battleship 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#pragma once
#include "Aircraft.h"
#include "Battleship.h"
#include "Cruiser.h"
#include "Destroyer.h"

// Defender 정의
class Defender {
    friend class GameManager; // GameManager 클래스가 Defender의 private 멤버 변수에 접근 가능하도록 함
private:
    Aircraft m_aircraft; // Aircraft 객체 생성
    Battleship m_battleship; // Battleship 객체 생성
    Cruiser m_cruiser; // Cruiser 객체 생성
    Destroyer m_destroyer; // Destroyer 객체 생성
    Destroyer m_destroyer2; // Destroyer 객체 생성
    int dMap[8][8]; // Defender 맵 배열
public:
    Defender(); // 생성자
    void locateShips(); // Defender가 가진 배를 모두 랜덤 배치함
};
```

Defender 클래스는 각각의 배와 그 배의 위치 정보를 소유하고 있는 플레이어이다.

게임이 시작될 때 locateShips 메소드를 호출하여 배를 배치한다.

변수와 메소드에 관한 설명은 주석을 보면 알 수 있다.

1-4. Attacker.h 헤더 파일

```
// C++ BattleShip 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#pragma once

// Attacker 정의
class Attacker {
    friend class GameManager; // GameManager 클래스가 Attacker의 private 멤버 변수에 접근 가능하도록 함
private:
    int aMap[8][8]; // Attacker 맵 배열
public:
    Attacker(); // 생성자
};
```

Attacker 는 Defender 의 배를 공격하는 플레이어이다.

Attacker 는 배를 소유할 필요가 없으며 Hit 과 Miss 상태를 표시하는 자신의 맵만 가지고 있다.

변수와 메소드에 관한 설명은 주석을 보면 알 수 있다.

1-5. Ships.h 헤더 파일

```
// C++ BattleShip 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#pragma once
#include <ctime>
#include <iostream>

// Ship 클래스 정의. 각 배의 공통된 속성을 정의한다.
class Ship{
protected:
    int size; // 배의 크기
    int row_location[5]; // 배의 위치를 저장할 배열
    int col_location[5]; // 배의 위치를 저장할 배열
    const int MAP_SIZE = 8; // 맵 크기
public:
    bool isDestroy(int[][8]); // 배의 파괴 여부를 리턴하는 함수
};
```

Ship 클래스는 Aircraft, Battleship, Cruiser, Destroyer 이 각각의 배들의 부모 클래스이다.

이들의 공통적인 속성을 정의하고 있다.

변수와 메소드에 관한 설명은 주석을 보면 알 수 있다.

1-6. Aircraft.h, Battleship.h, Cruiser.h, Destroyer.h 헤더 파일

```
#pragma once
#include "Ship.h"

// Aircraft 배 정의, Ship 클래스를 상속받음
class Aircraft : public Ship{
public:
    Aircraft(); // 생성자
    void locate(int [][][8]); // 배를 랜덤한 곳에 배치하는 함수
};

#pragma once
#include "Ship.h"

// Battleship 배 정의
class Battleship : public Ship{
public:
    Battleship(); // 생성자
    void locate(int [][][8]); // 배를 랜덤한 곳에 배치하는 함수
};

#pragma once
#include "Ship.h"

// Cruiser 배 정의
class Cruiser : public Ship{
public:
    Cruiser(); // 생성자
    void locate(int [][][8]); // 배를 랜덤한 곳에 배치하는 함수
};

#pragma once
#include "Ship.h"

// Destroyer 배 정의
class Destroyer : public Ship{
public:
    Destroyer(); // 생성자
    void locate(int [][][8]); // 배를 랜덤한 곳에 배치하는 함수
};
```

각각의 클래스는 Ship 클래스를 상속받아 Ship 의 멤버 변수와 메소드를 이용하고, 각각의 생성자와 위치를 결정하는 메소드를 가지고 있다.

2. 구현

2-1. GameBoard 구현

2-1-1. void init()

```
void GameBoard::init(){
    initscr();
    resize_term(29, 70); // 콘솔 사이즈 조정
    start_color(); // 색 사용
    init_pair(1, COLOR_WHITE, COLOR_BLUE); // 색 프리셋 저장
    init_pair(2, COLOR_WHITE, COLOR_RED); // 색 프리셋 저장
    init_pair(3, COLOR_WHITE, COLOR_MAGENTA); // 색 프리셋 저장
    border('*', '*', '*', '*', '*', '*', '*', '*'); // 테두리 지정
    mvprintw(1, 1, "BATTLESHIP GAME");
}
```

1. initscr()로 ncurses TUI 모드를 시작한다.
2. resize_term()으로 콘솔 사이즈를 적절히 조정한다.
3. start_color()로 색과 관련된 함수를 사용할 수 있게 한다.
4. init_pair()로 사용할 색 프리셋을 미리 결정해 둔다. 2 번째 인자는 글자 색, 3 번째 인자는 배경색이다.
5. border()로 콘솔 테두리를 적절히 꾸며준다.
6. mvprintw()로 콘솔에 메시지를 출력한다.

2-1-2. void render()

```
void GameBoard::render() {  
    drawDefenderMap(); // Defender 맵을 그림  
    drawAttackerMap(); // Attacker 맵을 그림  
    drawStatusWindow(); // Status 윈도우를 그림  
    drawInputWindow(); // Input 윈도우를 그림  
  
    // 실제 출력  
    refresh();  
    wrefresh(DEFENDER);  
    wrefresh(ATTACKER);  
    wrefresh(STATUS);  
    wrefresh(INPUT);  
    wrefresh(RESULT);  
}
```

1. 각각의 윈도우를 그리는 함수를 실행하고, refresh()를 통하여 실제 화면에 출력한다.

2-1-3. void drawDefenderMap(), void drawAttackerMap()

```
void GameBoard::drawDefenderMap(){
    DEFENDER = newwin(10, 10, 3, 5); // Defender 윈도우 생성
    wbkgd(DEFENDER, COLOR_PAIR(1));
    watttrn(DEFENDER, COLOR_PAIR(1));
    box(DEFENDER, 0, 0);
    mvwprintw(DEFENDER, 0, 1, "DEFENDER");

    // 행, 열 표현
    for(int i = 0; i < MAP_SIZE; i++){
        mvprintw(i+4, 4, "%c", 'A' + i);
        mvprintw(13, i+6, "%c", '1' + i);
    }
}
```

```
void GameBoard::drawAttackerMap(){
    ATTACKER = newwin(10, 10, 16, 5); // Attacker 윈도우 생성
    wbkgd(ATTACKER, COLOR_PAIR(1));
    watttrn(ATTACKER, COLOR_PAIR(1));
    box(ATTACKER, 0, 0);
    mvwprintw(ATTACKER, 0, 1, "ATTACKER");

    // 행, 열 표현
    for(int i = 0; i < MAP_SIZE; i++){
        mvprintw(i+17, 4, "%c", 'A' + i);
        mvprintw(26, i+6, "%c", '1' + i);
    }
}
```

1. DEFENDER, ATTACKER 윈도우를 생성한다.
2. wbkgd(), watttrn()을 이용하여 윈도우의 색을 설정한다.
3. box()를 이용하여 테두리를 그려준다.
4. mvwprintw()로 각 윈도우의 제목을 출력한다.
5. 윈도우 바깥에 A~H 까지 행, 1~8 까지 열을 나타내준다.

2-1-4. void drawStatusWindow(), void drawInputWindow()

```
void GameBoard::drawStatusWindow(){
    STATUS = newwin(8, 30, 3, 27); // Status 윈도우 생성
    wbkgd(STATUS, COLOR_PAIR(2));
    wattron(STATUS, COLOR_PAIR(2));
    box(STATUS, 0, 0);
    mvwprintw(STATUS, 0, 1, "STATUS");
}

void GameBoard::drawInputWindow(){
    INPUT = newwin(2, 20, 16, 27); // Input 윈도우 생성
    wbkgd(INPUT, COLOR_PAIR(3));
    wattron(INPUT, COLOR_PAIR(3));
    box(INPUT, 0, 0);
    mvwprintw(INPUT, 0, 1, "INPUT (EX: A3)");
    mvwprintw(INPUT, 1, 1, "Input position: ");

    RESULT = newwin(1, 20, 19, 27); // Result 윈도우 생성
    wbkgd(RESULT, COLOR_PAIR(3));
    wattron(RESULT, COLOR_PAIR(3));
}
```

1. STATUS, INPUT, RESULT 윈도우를 생성한다.
2. wbkgd(), wattron()을 이용하여 윈도우의 색을 설정한다.
3. box()를 이용하여 테두리를 그려준다.
4. mvwprintw()로 각 윈도우의 제목을 출력한다.

2-1-5. void refreshDefenderMap(int [][][8])

```
void GameBoard::refreshDefenderMap(int dMap[][8]){
    wclear(DEFENDER); // Defender 윈도우 내용 삭제
    box(DEFENDER, 0, 0);
    mvwprintw(DEFENDER, 0, 1, "DEFENDER");
    for(int i = 0; i < MAP_SIZE; i++){
        for(int j = 0; j < MAP_SIZE; j++){
            // Defender의 맵을 탐색하며 각각의 배를 출력
            if(dMap[i][j] == 0){
                mvwprintw(DEFENDER, i+1, j+1, ".");
            }
            else if(dMap[i][j] == 1){
                mvwprintw(DEFENDER, i+1, j+1, "A");
            }
            else if(dMap[i][j] == 2){
                mvwprintw(DEFENDER, i+1, j+1, "B");
            }
            else if(dMap[i][j] == 3){
                mvwprintw(DEFENDER, i+1, j+1, "C");
            }
            else if(dMap[i][j] == 4){
                mvwprintw(DEFENDER, i+1, j+1, "D");
            }
        }
    }
    wrefresh(DEFENDER);
}
```

1. Defender 의 맵을 출력하는 함수이다.

2. 정수형 맵 배열에 저장되어 있는 수가 0 이면 비어있는 곳, 1 이면 Aircraft, 2 이면 Battleship, 3 이면 Cruiser, 4 이면 Destroyer 를 출력한다.

2-1-6. void refreshAttackerMap(int [][][8])

```
void GameBoard::refreshAttackerMap(int aMap[][8]){
    wclear(ATTACKER); // Attacker 윈도우 내용 삭제
    box(ATTACKER, 0, 0);
    mvwprintw(ATTACKER, 0, 1, "ATTACKER");
    for(int i = 0; i < MAP_SIZE; i++){
        for(int j = 0; j < MAP_SIZE; j++){
            // Attacker 맵을 탐색하며 상태 출력
            if(aMap[i][j] == 0){
                mvwprintw(ATTACKER, i+1, j+1, ".");
            }
            else if(aMap[i][j] == -1){
                mvwprintw(ATTACKER, i+1, j+1, "M");
            }
            else if(aMap[i][j] == 1){
                mvwprintw(ATTACKER, i+1, j+1, "H");
            }
        }
    }
    wrefresh(ATTACKER);
}
```

1. Attacker 의 맵을 출력하는 함수이다.

2. 정수형 맵 배열에 저장되어 있는 수가 0 이면 비어있는 곳, -1 이면 M(Miss 로 확인된 지역), 1 이면 H(Hit 으로 확인된 지역)를 출력한다.

2-1-7. bool inputAttack(int [][][8], int [][][8])

```
bool GameBoard::inputAttack(int dMap[][8], int aMap[][8]){
    // 입력 받음
    keypad(INPUT, false);
    curs_set(2);
    echo();
    mvwscanw(INPUT, 1, 17, "%s", position);

    int row = position[0] - 'A';
    int col = (position[1] - '0') - 1;

    if((row >= 0 && row <= 7) && (col >= 0 && col <= 7)){ // 잘못된 입력 여부 확인
        bool isHit = checkHit(row, col, dMap); // Hit인지 아닌지 확인
        if(isHit){ // Hit일 경우
            refreshInput(position);
            mvwprintw(RESULT, 0, 4, "-");
            mvwprintw(RESULT, 0, 6, "HIT!");
            aMap[row][col] = 1; // Hit 상태 표시
            dMap[row][col] = -1; // Hit 상태 표시
        }
        else{ // Miss일 경우
            refreshInput(position);
            mvwprintw(RESULT, 0, 4, "-");
            if(dMap[row][col] == -1) { // 이미 Hit된 곳일 경우
                mvwprintw(RESULT, 0, 6, "Already HIT!");
            }
            else {
                mvwprintw(RESULT, 0, 6, "MISS!");
                aMap[row][col] = -1; // Miss 상태 표시
            }
        }
    }
    else{ // 잘못된 입력일 경우
        wclear(INPUT);
        wclear(RESULT);
        box(INPUT, 0, 0);
        mvwprintw(INPUT, 0, 1, "INPUT (EX: A3)");
        mvwprintw(INPUT, 1, 1, "Input position: ");
        mvwprintw(RESULT, 0, 1, "Invalid input");
        wrefresh(INPUT);
        wrefresh(RESULT);
        return false;
    }
    wrefresh(INPUT);
    wrefresh(RESULT);
    return true;
}
```

1. keypad() 함수에 false 인자를 줌으로써 특수키를 입력하지 못하게 한다.
2. curs_set() 함수로 커서 모양을 설정한다.
3. echo() 함수로 문자 입력 시 입력한 값을 화면에 보이게 한다.
4. mvwscanw()로 입력을 받아 position 변수에 저장한다.
5. row, col 변수에 "A2" 같은 형식으로 들어온 입력을 (ex. 'A' -> 0, '2' -> 1)로 변환시켜 저장한다
6. 범위에 올바른 입력이 들어왔는지 확인한다.

6-1. 올바른 입력이라면, Hit 인지 Miss 인지 여부를 확인한 후, 결과를 bool 형 변수에 저장한다.

6-1-1. 만약 Hit 이라면 input 윈도우를 새로 출력하고, result 윈도우에 Hit 을 출력한다.

6-1-2. Attacker 의 맵과 Defender 의 맵에 상태를 표시해준다.

6-1-3. 만약 Miss 라면 input 윈도우를 새로 출력하고, 이미 Hit 된 곳이라면 Already Hit, 아니라면 Miss 를 출력한다.

6-1-4. true 를 리턴한다.

6-2. 잘못된 입력이라면, Invalid input 을 출력하고 false 를 리턴한다.

2-1-8. bool checkHit(int, int, int[][8])

```
bool GameBoard::checkHit(int row, int col, int dMap[][8]){
    if(dMap[row][col] > 0) return true; // Hit이면 true 리턴
    else return false;
}
```

1. Defender 의 맵의 값이 0 이상이라면 배가 정상적으로 존재하는 경우이므로 Hit 이다. true 를 리턴한다.
2. 0 미만이라면 파괴되었거나, 배가 존재하지 않는 곳이므로 Miss 이다. false 를 리턴한다.

2-1-9. void refreshInput(const char*)

```
void GameBoard::refreshInput(const char* position){
    wclear(INPUT);
    wclear(RESULT);
    box(INPUT, 0, 0);
    mvwprintw(INPUT, 0, 1, "INPUT (EX: A3)");
    mvwprintw(INPUT, 1, 1, "Input position: ");
    mvwprintw(RESULT, 0, 1, position);
}
```

1. Input 윈도우와 result 윈도우를 wclear()를 이용하여 지운다.
2. 다시 그려준다. Result 윈도우에는 사용자가 입력한 좌표를 출력한다.

2-1-10. void refreshStatus(int, bool, bool, bool, bool, bool)

```
void GameBoard::refreshStatus(int turn, bool aircraft_destroy, bool battleship_destroy, bool cruiser_destroy, bool destroyer_destroy, bool destroyer2_destroy){
    wclear(STATUS);
    box(STATUS, 0, 0);
    mvwprintw(STATUS, 0, 1, "STATUS");
    mvwprintw(STATUS, 1, 1, "TURN: ");
    mvwprintw(STATUS, 1, 7, "%d", turn);

    mvwprintw(STATUS, 2, 1, "AIRCRAFT: ");
    if(!aircraft_destroy) { mvwprintw(STATUS, 2, 11, "AAAAA"); } // 파괴되지 않은 경우
    else { mvwprintw(STATUS, 2, 11, "DESTROYED"); } // 파괴된 경우

    mvwprintw(STATUS, 3, 1, "BATTLESHIP: ");
    if(!battleship_destroy) { mvwprintw(STATUS, 3, 13, "BBBB"); } // 파괴되지 않은 경우
    else { mvwprintw(STATUS, 3, 13, "DESTROYED"); } // 파괴된 경우

    mvwprintw(STATUS, 4, 1, "CRUISER: ");
    if(!cruiser_destroy) { mvwprintw(STATUS, 4, 10, "CCC"); } // 파괴되지 않은 경우
    else { mvwprintw(STATUS, 4, 10, "DESTROYED"); } // 파괴된 경우

    mvwprintw(STATUS, 5, 1, "DESTROYER 1: ");
    if(!destroyer_destroy) { mvwprintw(STATUS, 5, 14, "DD"); } // 파괴되지 않은 경우
    else { mvwprintw(STATUS, 5, 14, "DESTROYED"); } // 파괴된 경우

    mvwprintw(STATUS, 6, 1, "DESTROYER 2: ");
    if(!destroyer2_destroy) { mvwprintw(STATUS, 6, 14, "DD"); } // 파괴되지 않은 경우
    else { mvwprintw(STATUS, 6, 14, "DESTROYED"); } // 파괴된 경우

    wrefresh(STATUS);
}
```

1. wclear()를 이용하여 Status 윈도우를 지운다.
2. 다시 그려준다.
3. 현재 진행된 턴의 수와 각 배의 파괴 상태를 조건에 따라 나타내준다.

2-1-11. void exitGame()

```
// 게임 종료
void GameBoard::exitGame(){
    mvprintw(15, 5, "GAME END - Press ANY KEY to Exit Game."); // 게임 종료 메세지 출력
    refresh();
    getch();
    delwin(DEFENDER);
    delwin(ATTACKER);
    delwin(STATUS);
    delwin(INPUT);
    delwin(RESULT);
    endwin();
}
```

1. 게임을 종료하는 함수.
2. 키 입력을 받으면 모든 윈도우를 삭제하고 종료한다.

2-2. GameManager 구현

2-2-1. GameManager()

```
GameManager::GameManager(){  
    turn = 0; // 현재 턴을 0으로 초기화  
}
```

1. GameManager 클래스가 관리하는 멤버 변수인 turn 을 초기화 한다.

2-2-2. void play()

```
void GameManager::play(){  
    m_gameboard.init(); // 기초 세팅 초기화  
    m_gameboard.render(); // 화면에 게임판 출력  
    m_defender.locateShips(); // 수비자는 배를 랜덤한 곳에 배치  
    m_gameboard.refreshDefenderMap(m_defender.dMap); // Defender 맵 초기화  
    m_gameboard.refreshAttackerMap(m_attacker.aMap); // Attacker 맵 초기화  
    m_gameboard.refreshStatus(0, false, false, false, false, false); // Status 초기화  
    while(!allDestroy()){ // 모든 배가 파괴될때 까지 게임 진행  
        bool isCorrect = m_gameboard.inputAttack(m_defender.dMap, m_attacker.aMap); // 좌표 input  
        if(isCorrect) turn++; // 잘못된 입력이 아니면 턴 수 증가  
  
        // 변경된 상태 refresh  
        m_gameboard.refreshStatus(turn,  
            m_defender.m_aircraft.isDestroy(m_attacker.aMap), m_defender.m_battleship.isDestroy(m_attacker.aMap),  
            m_defender.m_cruiser.isDestroy(m_attacker.aMap), m_defender.m_destroyer.isDestroy(m_attacker.aMap),  
            m_defender.m_destroyer2.isDestroy(m_attacker.aMap) );  
  
        m_gameboard.refreshAttackerMap(m_attacker.aMap); // 공격자 맵 refresh  
    }  
    m_gameboard.exitGame(); // 게임 종료  
}
```

1. GameBoard 객체의 메소드를 호출하여 게임판을 출력한다.
2. Defender 객체의 locateShips()를 호출하여 배를 랜덤한 곳에 배치한다.
3. 초기 상태의 맵과 상태 창을 출력한다.
4. 배가 모두 파괴되기 전까지 아래를 반복한다.
 - 좌표를 입력받는다.
 - 입력에 따라 변경된 상태를 GameBoard 객체에 넘겨주어 상태 창을 새로 그린다.
 - 입력에 따라 변경된 상태를 GameBoard 객체에 넘겨주어 Attacker 맵을 새로 그린다.
5. 반복이 끝나면 게임을 종료한다.

2-2-3. bool allDestroy()

```
bool GameManager::allDestroy(){
    // 각각의 배가 모두 파괴되었는지 확인
    if(m_defender.m_aircraft.isDestroy(m_attacker.aMap) && m_defender.m_battleship.isDestroy(m_attacker.aMap)
    && m_defender.m_cruiser.isDestroy(m_attacker.aMap) && m_defender.m_destroyer.isDestroy(m_attacker.aMap)
    && m_defender.m_destroyer2.isDestroy(m_attacker.aMap)){
        return true;
    }
    else return false;
}
```

1. 각각의 배 객체가 가지고 있는 isDestroy() 메소드를 통해 파괴되었는지 여부를 확인하고, 모두 파괴되었다면 true 를 리턴한다.
2. 아니면 false 를 리턴한다.

2-3. Ship 구현

2-3-1. bool isDestroy(int [][][8])

```
// 모두 HIT 상태면 true, 아니면 false 리턴
bool Ship::isDestroy(int aMap[][8]){
    for(int i = 0; i < size; i++){
        if(aMap[row_location[i]][col_location[i]] != 1){
            return false;
        }
    }
    return true;
}
```

1. row_location 과 col_location 은 배의 좌표가 저장되어 있는 배열이다.
2. 이곳의 값을 가져와 Attacker 의 맵을 각 배의 size 만큼 검사하여 모두 Hit 된 상태(배열의 값이 1 인 상태)이면 true, 아니면 false 를 리턴한다.

2-4 Aircraft 구현

2-4-1. Aircraft()

```
// 멤버 변수 초기화
Aircraft::Aircraft(){
    size = 5;
    for(int i = 0; i < size; i++){
        row_location[i] = -1;
        col_location[i] = -1;
    }
}
```

1. Aircraft 의 크기를 초기화한다.
2. 좌표를 저장하는 배열을 초기화한다.

2-4-2. void locate(int[][8])

```
void Aircraft::locate(int dMap[][8]){
    srand((unsigned int)time(NULL)); // 랜덤 시드
    bool isVertical; // 가로 배치, 세로 배치 여부 확인 변수
    int locateType = rand() % 100 + 1; // 가로, 세로 배치 랜덤 결정
    if(locateType % 2 == 0) { isVertical = false; } // 가로 배치
    else { isVertical = true; } // 세로 배치

    int row = rand() % MAP_SIZE; // 0~7 값이 랜덤으로 결정
    int col = rand() % MAP_SIZE; // 0~7 값이 랜덤으로 결정
    int idx = 0; // 위치 저장 배열의 인덱스

    // 가로 배치
    if(!isVertical){
        // 오른쪽으로 배치
        if(col <= 3){
            for(int i = col; i < col + size; i++){
                dMap[row][i] = 1; // Defender 맵 배열에 저장
                row_location[idx] = row;
                col_location[idx] = i;
                idx++;
            }
        }
        // 왼쪽으로 배치
        else{
            for(int i = col; i > col - size; i--){
                dMap[row][i] = 1; // Defender 맵 배열에 저장
                row_location[idx] = row;
                col_location[idx] = i;
                idx++;
            }
        }
    }

    // 세로 배치
    else{
        // 아래쪽으로 배치
        if(row <= 3){
            for(int i = row; i < row + size; i++){
                dMap[i][col] = 1; // Defender 맵 배열에 저장
                row_location[idx] = i;
                col_location[idx] = col;
                idx++;
            }
        }
        // 위쪽으로 배치
        else{
            for(int i = row; i > row - size; i--){
                dMap[i][col] = 1; // Defender 맵 배열에 저장
                row_location[idx] = i;
                col_location[idx] = col;
                idx++;
            }
        }
    }
}
```

1. 랜덤 시드를 결정한다.
2. 배를 가로로 배치할 지, 세로로 배치할 지에 대한 여부를 랜덤하게 결정한다.
3. 그 후 배가 위치할 하나의 좌표를 랜덤으로 결정한다.
4. Aircraft 는 가장 처음으로 배치될 배이기 때문에, 다른 배가 위치해 있는지에 대한 여부는 고려할 필요가 없다.
5. 배를 가로로 배치한다면,
 - 5-1. [0 1 2 3 4 5 6 7] 열에서 3 열 이하라면 왼쪽 방향으로 5 칸을 할당할 수 없으므로 오른쪽으로 배치한다.
 - 5-2. [0 1 2 3 4 5 6 7] 열에서 4 열 이상이라면 오른쪽 방향으로 5 칸을 할당할 수 없으므로 왼쪽으로 배치한다.
6. 배를 세로로 배치한다면,
 - 6-1. [0 1 2 3 4 5 6 7] 행에서 3 행 이하라면 위쪽 방향으로 5 칸을 할당할 수 없으므로 아래쪽으로 배치한다.
 - 6-2. [0 1 2 3 4 5 6 7] 행에서 4 행 이상이라면 아래쪽 방향으로 5 칸을 할당할 수 없으므로 위쪽으로 배치한다.
7. 배가 배치된 좌표를 배열에 저장한다.

2-5. Battleship, Cruiser, Destroyer 구현

2-5-1. 생성자

```
// 멤버 변수 초기화
Battleship::Battleship(){
    size = 4;
    for(int i = 0; i < size; i++){
        row_location[i] = -1;
        col_location[i] = -1;
    }
}
```

```
Destroyer::Destroyer(){
    size = 2;
    for(int i = 0; i < size; i++){
        row_location[i] = -1;
        col_location[i] = -1;
    }
}
```

```
// 멤버 변수 초기화
Cruiser::Cruiser(){
    size = 3;
    for(int i = 0; i < size; i++){
        row_location[i] = -1;
        col_location[i] = -1;
    }
}
```

1. 각 배의 크기를 초기화한다.
2. 좌표를 저장하는 배열을 초기화한다.

2-5-2. void locate(int[][8])

Aircraft 를 제외한 나머지 배들은 로직이 모두 같으므로 Battleship 만 대표로 설명을 작성한다.

```
void Battleship::locate(int dMap[][8]){
    srand((unsigned int)time(NULL)); // 랜덤 시드
    bool isVertical; // 가로 배치, 세로 배치 여부 확인 변수
    int row;
    int col;
    int idx = 0;

    while(true){
        int locateType = rand() % 100 + 1; // 가로, 세로 배치 랜덤 결정
        if(locateType % 2 == 0) { isVertical = false; } // 가로 배치
        else { isVertical = true; } // 세로 배치

        row = rand() % MAP_SIZE; // 0~7 값이 랜덤으로 결정
        col = rand() % MAP_SIZE; // 0~7 값이 랜덤으로 결정

        if(dMap[row][col] > 0) continue; // 최초로 선택된 좌표에 이미 배가 있다면 좌표를 다시 결정함

        // 가로 배치
        if(!isVertical){
            if(col <= 2){
                // 오른쪽
                if(dMap[row][col] == 0 && dMap[row][col+1] == 0 && dMap[row][col+2] == 0 && dMap[row][col+3] == 0){
                    for(int i = col; i < col + size; i++){
                        dMap[row][i] = 2; // Defender 맵 배열에 저장
                        row_location[idx] = row;
                        col_location[idx] = i;
                        idx++;
                    }
                    break;
                }
            }
            else continue;
        }

        // 세로 배치
        if(isVertical){
            if(row <= 2){
                // 아래쪽
                if(dMap[row][col] == 0 && dMap[row+1][col] == 0 && dMap[row+2][col] == 0 && dMap[row+3][col] == 0){
                    for(int i = row; i < row + size; i++){
                        dMap[i][col] = 2; // Defender 맵 배열에 저장
                        row_location[idx] = i;
                        col_location[idx] = col;
                        idx++;
                    }
                    break;
                }
            }
            else continue;
        }
    }
}
```

```
else if(col <= 4){
    // 오른쪽
    if(dMap[row][col] == 0 && dMap[row][col+1] == 0 && dMap[row][col+2] == 0 && dMap[row][col+3] == 0){
        for(int i = col; i < col + size; i++){
            dMap[row][i] = 2; // Defender 맵 배열에 저장
            row_location[idx] = row;
            col_location[idx] = i;
            idx++;
        }
        break;
    }

    // 왼쪽
    else if(dMap[row][col] == 0 && dMap[row][col-1] == 0 && dMap[row][col-2] == 0 && dMap[row][col-3] == 0){
        for(int i = col; i > col - size; i--){
            dMap[row][i] = 2; // Defender 맵 배열에 저장
            row_location[idx] = row;
            col_location[idx] = i;
            idx++;
        }
        break;
    }
    else continue;
}
```

```

else{
    // 왼쪽
    if(dMap[row][col] == 0 && dMap[row][col-1] == 0 && dMap[row][col-2] == 0 && dMap[row][col-3] == 0){
        for(int i = col; i > col - size; i--){
            dMap[row][i] = 2; // Defender 맵 배열에 저장
            row_location[idx] = row;
            col_location[idx] = i;
            idx++;
        }
        break;
    }
    else continue;
}
}
}

```

1. 랜덤 시드를 결정한다.

2. 배의 배치가 완료될 때까지 아래 항목을 반복한다.

- 가로, 세로 배치 여부를 랜덤으로 결정한다.
- 배가 위치할 하나의 좌표를 랜덤으로 결정한다.
- 만약 최초로 선택된 좌표에 이미 배가 있다면 다시 처음으로 돌아간다.
- 가로 배치일 경우,
 - [0 1 2 3 4 5 6 7] 열에서 2 열 이하라면 왼쪽 방향으로 4 칸을 할당할 수 없으므로, 오른쪽으로 배치한다. 이 때, 오른쪽 3 칸에 배가 존재하는 지 확인한다. 존재한다면 다시 처음으로 돌아간다.
 - [0 1 2 3 4 5 6 7] 열에서 3 열 혹은 4 열이라면 오른쪽, 왼쪽 모두 4 칸을 할당할 수 있으므로, 둘 중 배를 배치할 수 있는 방향으로 배를 배치한다. 만약 두 방향 다 배치가 가능하다면, 오른쪽 방향으로 배치한다.
 - 5 열 이상이라면 오른쪽 방향으로 4 칸을 할당할 수 없으므로, 왼쪽으로 배치한다. 이 때, 왼쪽 3 칸에 배가 존재하는지 확인한다. 존재한다면 다시 처음으로 돌아간다.
- (세로 배치일 경우와 가로 배치일 경우의 로직이 일치하므로 가로 배치일 경우만 소개한다.)

2-6. Defender 구현

2-6-1. Defender()

```
// Defender 맵 배열 초기화
Defender::Defender(){
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 8; j++){
            dMap[i][j] = 0;
        }
    }
}
```

1. Defender의 맵 배열을 0(아무것도 놓여있지 않음)으로 초기화한다.
2. -1: 배가 Hit된 상태, 1: Aircraft, 2: Battleship, 3: Cruiser, 4: Destroyer를 나타낸다.

2-6-2. void locateShips()

```
void Defender::locateShips(){
    m_aircraft.locate(dMap); // Aircraft 배치
    m_battleship.locate(dMap); // Battleship 배치
    m_cruiser.locate(dMap); // Cruiser 배치
    m_destroyer.locate(dMap); // Destroyer 배치
    m_destroyer2.locate(dMap); // Destroyer 배치
}
```

1. 각각의 객체에 구현되어 있는 배를 배치하는 메소드를 호출한다.

2-7. Attacker 구현

2-7-1. Attacker()

```
// Attacker 맵 배열 초기화
Attacker::Attacker(){
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 8; j++){
            aMap[i][j] = 0;
        }
    }
}
```

1. Attacker의 맵 배열을 0(아직 확인하지 않음) 상태로 초기화 한다.
2. -1: Miss, 1: Hit 을 나타낸다.

2-8. Main

```
// C++ BattleShip 프로젝트
// 작성 일자: 2018-05-31
// 학번 : 20171676
// 이름 : 이정우
#include "GameManager.h"

using namespace std;

int main(){
    GameManager game;
    game.play();
    return 0;
}
```

3. 결과물

```
ljwt@DESKTOP-9LSM6G2: /mnt/c/cpp-workspace/BattleShip
*****
*BATTLESHIP GAME
*
* DEFENDER
* A .....
* B .AAAAA..
* C ...DD...
* D ..D...B
* E ..D...B
* F ...C..B
* G ...C..B
* H ...C...
* 12345678
*
* STATUS
* TURN: 0
* AIRCRAFT: AAAAA
* BATTLESHIP: BBBB
* CRUISER: CCC
* DESTROYER 1: DD
* DESTROYER 2: DD
*
* ATTACKER
* A .....
* B .....
* C .....
* D .....
* E .....
* F .....
* G .....
* H .....
* 12345678
*
* INPUT (EX: A3)
* Input position:
*
*****
```

초기화면

```
ljwt@DESKTOP-9LSM6G2: /mnt/c/cpp-workspace/BattleShip
*****
*BATTLESHIP GAME
*
* DEFENDER
* A .....
* B .AAAAA..
* C ...DD...
* D ..D...B
* E ..D...B
* F ...C..B
* G ...C..B
* H ...C...
* 12345678
*
* STATUS
* TURN: 9
* AIRCRAFT: DESTROYED
* BATTLESHIP: BBBB
* CRUISER: CCC
* DESTROYER 1: DD
* DESTROYER 2: DD
*
* ATTACKER
* A .....
* B .HHHHMM
* C .M.....
* D .M.....
* E .....
* F .....
* G .....
* H .....
* 12345678
*
* INPUT (EX: A3)
* Input position:
*
* D2 - MISS!
*
*****
```

플레이 중

```
lhw@DESKTOP-9LSM6G2: /mnt/c/cpp-workspace/BattleShip
*****
*BATTLESHIP GAME
*
* DEFENDER
* A .....
* B .AAAAA..
* C ..DD...
* D ..D...B
* E ..D...B
* F ....C..B
* G ....C..B
* H ....C...
*
* 12345678
*
* STATUS
* TURN: 21
* AIRCRAFT: DESTROYED
* BATTLESHIP: DESTROYED
* CRUISER: DESTROYED
* DESTROYER 1: DESTROYED
* DESTROYER 2: DESTROYED
*
* GAME END - Press ANY KEY to Exit Game.
* ATTACKER
* A .....
* B .HHHHMM
* C M..HH...
* D .MH...H
* E .H...H
* F ....HM.H
* G ....H..H
* H ....H...
*
* 12345678
*
* INPUT (EX: A3)
* Input position:
*
* C5 - HIT!
*****
```

게임이 끝났을 때