배틀쉽 프로젝트 보고서

C++ 프로그래밍 05분반

20171701

정지현

MAIN.CPP

- main함수를 담고 있는 파일.
- BattleShipApp 객체를 하나 생성하여 그 객체의 Play 함수를 실행한다.

```
#include "BattleShipApp.h"

int main(){
    BattleShipApp battleShip = BattleShipApp();
    battleShip.Play();
    return 0;
}
```

BATTLESHIPAPP.H

- 게임을 실행하는 Play함수
- 색깔을 정의하고 기본적인 윈도우를 생성하는 Init 함수
- 배를 배치하는 arrangeShips 함수
- 배를 배치할 때 놓을 수 있는 위치인지 확인하는 isSafe 함수
- 생성한 윈도우들과 배들을 화면에 표시하는 Render 함수
- 게임을 끝내는 Destory함수
- 게임이 끝났는지 확인하는 isFinished 함수
- 글자를 입력받고, 공격을 수행하여 결과를 보여주는 gamePlay 함수

```
class BattleShipApp{
    public:
        BattleShipApp();
        ~BattleShipApp();

        void Play();

    protected:
        void Init();
        void Render();
        void Destroy();
        bool isFinished();
        void gamePlay();
        void arrangeShips();
        bool isSafe(int x, int y, bool direction, int size);

    protected:
        BattleShipMap* m_pMap;
        Player* m_playerMap;
        StatPane* m_pStatPane;
        InputPane* m_pInputPane;
        Aircraft* aircraft;
        BattleShip* battleship;
        Cruiser* cruiser;
        Destroyer* destroyer_one;
        Destroyer* destroyer_two;
};
```

BATTLESHIPAPP.CPP - INIT()

- 스크린을 생성하고, 스크린에 입력받은 문자 출력하지 않도록, 커서도 출력하지 않도록 한다.

- 컬러 페어를 만들고, 윈도우들을 생성한다.

```
void BattleShipApp::Init(){
    initscr();
    start_color();
    cbreak();
    refresh();
    noecho();
    curs_set(0);

    init_pair(1, COLOR_GREEN, COLOR_BLACK);
    init_pair(2, COLOR_CYAN, COLOR_BLACK);
    init_pair(3, COLOR_YELLOW, COLOR_BLACK);
    init_pair(4, COLOR_WHITE, COLOR_RED);
    init_pair(5, COLOR_WHITE, COLOR_YELLOW);
    init_pair(6, COLOR_WHITE, COLOR_MAGENTA);
    init_pair(7, COLOR_WHITE, COLOR_CYAN);
    init_pair(8, COLOR_RED, COLOR_BLACK);

    m_pMap = new BattleShipMap();
    m_playerMap = new Player();
    m_pStatPane = new StatPane(30, 5, 30, 7);
    m_pInputPane = new InputPane(30, 17, 30, 5);

}
```

BATTLESHIPAPP.CPP - ARRANGESHIPS()

- 랜덤을 이용한다.
- 방향은 true or false
  - true는 가로, false는 세로
- 길이를 감안하여 랜덤 좌표를 생성한다.
- 다른 배들에 대해서도 기본적인 알고리즘은 같다.

```
void BattleShipApp::arrangeShips(){
    srand((unsigned int)time(NULL));
    //Aircraft
    bool dir = rand() % 2;
    int a = rand() % 4;
    int b = rand() % 8;
    if (dir){
        aircraft = new Aircraft(b, a, dir);
        m_pMap -> update(b, a, dir, 5, 'A');
    }else{
        aircraft = new Aircraft(a, b, dir);
        m_pMap -> update(a, b, dir, 5, 'A');
    }
```

BATTLESHIPAPP.CPP - ISSAFE()

- 디펜더 역할을 하는 BattleShipMap의 isSafe 함수를 호출하여 판별한다.

```
bool BattleShipApp::isSafe(int x, int y, bool direction, int size){
    if (m_pMap -> isSafe(x, y, direction, size)) return true;
    else return false;
}
```

BATTLESHIPAPP.CPP - PLAY()

- 필요한 함수를 호출한다.
- 생성 - 그리기 - 게임 진행 - 게임 끝 순으로 진행된다.

```
void BattleShipApp::Play(){
    Init();
    arrangeShips();
    Render();
    gamePlay();
    Destroy();
}
```

BATTLESHIPAPP.CPP - RENDER()

- 생성한 윈도우들과 배들의 Draw함수를 호출한다.

```
void BattleShipApp::Render(){
    mvprintw(1, 1, "<< Battle Ship Game >>");

    m_pMap -> Draw();
    m_playerMap -> Draw();
    m_pStatPane -> Draw(0, 0, 0, 0);
    m_pInputPane -> Draw();

    refresh();
    aircraft -> Draw(m_pMap -> getWindow());
    battleship -> Draw(m_pMap -> getWindow());
    cruiser -> Draw(m_pMap -> getWindow());
    destroyer_one -> Draw(m_pMap -> getWindow());
    destroyer_two -> Draw(m_pMap -> getWindow());
}
```

BATTLESHIPAPP.CPP - ISFINISHED()

- 모든 배가 파괴되면 true를 리턴한다.

```
bool BattleShipApp::isFinished(){
    return aircraft -> isDestroyed() && battleship -> isDestroyed() &&
    cruiser -> isDestroyed() && destroyer_one -> isDestroyed() &&
    destroyer_two -> isDestroyed();
}
```

BATTLESHIPAPP.CPP - GAMEPLAY() - 1

- 게임이 끝나지 않은 동안 반복한다.
- 첫번째는 A - H, a - h 의 알파벳만 입력받는다.
- 두번째는 1 - 8 의 숫자만 입력받는다.
- 첫번째 원소를 입력받은 후, Draw함수를 이용하여 그린다.

```
void BattleShipApp::gamePlay(){
    while (!isFinished()){
        char row, col;
        while (1){
            char temp = getch();
            if (int(temp) >= 65 && int(temp) <= 72){
                row = temp;
                break;
            }
            if (int(temp) >= 97 && int(temp) <= 104){
                row = temp - 32;
                break;
            }
        }
        m_pInputPane -> Draw();
        m_pInputPane -> Draw(row);
        while(1){
            char temp = getch();
            if (int(temp) >= 49 && int(temp) <= 56){
                col = temp;
                break;
            }
        }
```

BATTLESHIPAPP.CPP - GAMEPLAY() - 2

- 공격을 수행한다.
- attack 함수의 return 값은 string 인데, 그 결과에 따라서 다른 로직을 실행하게 된다.
- hit인 경우는 플레이어 맵에 공격을 한번 더 수행하고, 그것도 hit을 리턴한다면 그 부분의 데이터에 따라서 무슨 배인지 파악한 후에, 공격을 수행한 수 파괴되었는지 확인한다.
- 파괴되었으면 Destroyed를 출력하고, 아니라면 hit을 출력한다.
- 이미 공격한 부분이라면 try-again을 출력한다.

```
std::string s = m_pMap -> attack((int)row - 65, (int)col - 49);
if (s == "hit"){
    std::string p = m_playerMap -> attack((int)row - 65, (int)col - 49);
    if (p == "hit"){
        char c = m_pMap -> getData((int)row - 65, (int)col - 49);
        char temp;
        switch (c){
            case 'A':
                if (!aircraft-> isDestroyed()){
                    aircraft -> attacked();
                    m_pStatPane -> turnPass();
                    m_playerMap -> update((int)row - 65, (int)col - 49, 'H');
                    if (aircraft -> isDestroyed()){
                        aircraft -> Draw(m_playerMap -> getWindow());
                        m_pInputPane -> Draw(row, col, "Aircraft Destroyed");
                    }else{
                        m_pInputPane -> Draw(row, col, s);
                    }
                }
                break;
```

```
}else{
    m_pInputPane -> Draw(row, col, "try-again");
}
```

BATTLESHIPAPP.CPP - GAMEPLAY() - 3

- 만약 리턴이 miss인 경우, 이미 플레이어 맵에서 miss였다고 표기된 경우 다시 입력하라고 표시한다.

- 아니라면 miss이므로 턴을 하나 늘리고 miss라고 표기한다.

```
else{
    if (m_playerMap -> getData(int(row) - 65, int(col) - 49) == 'M'){
        m_pInputPane -> Draw(row, col, "try-again");
    }else{
        m_pStatPane -> turnPass();
        m_playerMap -> update((int)row - 65, (int)col - 49, 'M');
        m_pInputPane -> Draw(row, col, s);
    }
}
```

BATTLESHIPAPP.CPP - GAMEPLAY() - 4

- 배가 만약 파괴되었다면 플레이어맵에도 디펜더맵과 같이 표시해준다.

- 그리고 상태를 표시하는 부분에도 파괴된 배는 빨간색으로 바꿔서 표기한다.

```
if (aircraft -> isDestroyed()){
    aircraft -> Draw(m_playerMap -> getWindow());
}
if (battleship -> isDestroyed()){
    battleship -> Draw(m_playerMap -> getWindow());
}
if (cruiser -> isDestroyed()){
    cruiser -> Draw(m_playerMap -> getWindow());
}
if (destroyer_one -> isDestroyed()){
    destroyer_one -> Draw(m_playerMap -> getWindow());
}
if (destroyer_two -> isDestroyed()){
    destroyer_two -> Draw(m_playerMap -> getWindow());
}
m_pStatPane -> Draw(aircraft -> isDestroyed(), battleship -> isDestroyed(),
cruiser -> isDestroyed(), destroyer_one -> isDestroyed() && destroyer_two -> isDestroyed());
```

BATTLESHIPAPP.CPP - DESTROY()

- 게임이 끝난 경우, 키를 하나 입력받으면 G G Good Game! 을 출력한다.

- 그 다음 키를 하나 더 입력받으면, 게임을 종료한다.

```
void BattleShipApp::Destroy(){
    getch();
    m_pInputPane -> Draw();
    m_pInputPane -> Draw('G', 'G', "Good Game!");
    getch();
    endwin();
    delete m_pMap;
}
```

PANE.H

- 기본적인 윈도우들을 정의한다. 가로, 세로 크기와 시작 좌표를 정의한다.
- 윈도우를 그리는 Draw함수
- WINDOW 포인터를 리턴하는 getWindow 함수로 구성된다.

```
class Pane{
    public:
        Pane(int x, int y, int width, int height);
        virtual ~Pane();

        virtual void Draw();

        WINDOW* getWindow();

    protected:
        int m_width, m_height;
        int m_x, m_y;
        WINDOW* m_pWindow;
};
```

PANE.CPP

- 생성자는 인자로 좌표와 너비, 높이를 받아서 초기화해주고, 새로운 윈도우를 생성한다.

- getWindow함수는 현재 객체의 윈도우를 리턴해준다.

```
Pane::Pane(int x, int y, int width, int height)
    :m_x(x), m_y(y), m_width(width), m_height(height)
{
    m_pWindow = newwin(height, width, y, x);
    box(m_pWindow, 0, 0);
    wrefresh(m_pWindow);
}

Pane::~Pane(){
    delwin(m_pWindow);
}

void Pane::Draw(){
    box(m_pWindow, 0, 0);
    wrefresh(m_pWindow);
}

WINDOW* Pane::getWindow(){
    return m_pWindow;
}
```

BATTLESHIPMAP.H

- 맵의 크기를 지정하고, 함수들을 정의한다.

- 배들의 정보를 담고 있는 m_mapData 배열을 가지고 있다.

```
#include "Pane.h"
#include <string>

#define MAP_SIZE 8

class BattleShipMap : public Pane{
    public:
        BattleShipMap();
        ~BattleShipMap();
        void update(int row, int col, bool direction, int size, char name);
        void Draw();
        bool isSafe(int a, int b, bool direction, int size);
        std::string attack(int col, int row);
        char getData(int col, int row);

    protected:
        char m_mapData[MAP_SIZE][MAP_SIZE];
};
```

BATTLESHIPMAP.CPP
- 생성자, 소멸자

- m_mapData를 모두 0으로 초기화한다.
- 그리고 옆쪽에 A - H, 1 - 8을 표시한다.
- 제목은 DEFENDER라고 달아준다.

```
BattleShipMap::BattleShipMap()
    :Pane(4, 4, MAP_SIZE + 3, MAP_SIZE + 2)
{
    for(int i = 0;i < MAP_SIZE;i++){
        for(int j = 0;j < MAP_SIZE;++j){
            m_mapData[i][j] = '0';
        }
    }

    for(int i = 0;i < MAP_SIZE;++i){
        mvprintw(i + 1 + m_y, m_x - 1, "%c", 'A' + i);
        mvprintw(m_y + m_height, m_x + 2 + i, "%d", 1 + i);
    }

    mvwprintw(m_pWindow, 0, 2, "DEFENDER");
}

BattleShipMap::~BattleShipMap(){

}
```

BATTLESHIPMAP.CPP - DRAW()

- 데이터가 0이면 -, 아니라면 배열의 데이터를 출력한다.

```
void BattleShipMap::Draw(){
    wattron(m_pWindow, COLOR_PAIR(1));
    for(int i = 0;i < MAP_SIZE;++i){
        for(int j = 0;j < MAP_SIZE;++j){
            if (m_mapData[i][j] == '0')
                mvwprintw(m_pWindow, i + 1, j + 2, "-");
            else{
                mvwprintw(m_pWindow, i + 1, j + 2, "%c", m_mapData[i][j]);
            }
        }
    }
    wattroff(m_pWindow, COLOR_PAIR(1));

    wrefresh(m_pWindow);
}
```

BATTLESHIPMAP.CPP - UPDATE()

- 배를 생성한 후에, 배열에 배의 데이터를 업데이트할 때 사용한다.

- 가로 세로를 구분하여 배 이름의 첫 글자를 채운다.

```
void BattleShipMap::update(int row, int col, bool direction, int size, char name){
    if (direction){
        for(int i = 0;i < size;i++){
            m_mapData[row][col + i] = name;
        }
    }else{
        for(int i = 0;i < size;i++){
            m_mapData[row + i][col] = name;
        }
    }
}
```

BATTLESHIPMAP.CPP - ISSAFE()

- 배를 배치하는 범위 내에 이미 배가 배치되어 있는지 확인한다.

```
bool BattleShipMap::isSafe(int a, int b, bool dir, int size){
    if (dir){
        for(int i = 0;i < size;i++){
            if (m_mapData[a][b + i] != '0') return false;
        }
    }else{
        for(int i = 0;i < size;i++){
            if (m_mapData[a + i][b] != '0') return false;
        }
    }
    return true;
}
```

BATTLESHIPMAP.CPP - GETDATA()

- 배열의 현재 위치에 있는 데이터를 리턴한다.

```
char BattleShipMap::getData(int row, int col){
    return m_mapData[row][col];
}
```

BATTLESHIPMAP.CPP - ATTACK()

- 인자로 받아온 좌표에 배가 위치한다면 hit, 아니면 miss 를 리턴한다.

```
std::string BattleShipMap::attack(int row, int col){
    if (m_mapData[row][col] == '0'){
        return "miss";
    }
    else{
        return "hit";
    }
}
```

PLAYER.H

- attacker를 정의한다.
- battleshipmap과 비슷한 구조를 가진다.

```
#include <string>
#include "Pane.h"

#define MAP_SIZE 8

class Player : public Pane{
    public:
        Player();
        ~Player();

        void Draw();
        void update(int row, int col, char data);
        std::string attack(int row, int col);
        char getData(int row, int col);

    protected:
        char m_mapData[MAP_SIZE][MAP_SIZE];
};
```

PLAYER.CPP
- 생성자, 소멸자

- battleshipmap과 기본적인 구조는 같다.

- 그리는 좌표가 다르고, ATTACKER로 표시된다.

```
Player::Player()
    :Pane(4, 16, MAP_SIZE + 3, MAP_SIZE + 2)
{
    for(int i = 0;i < MAP_SIZE;i++){
        for(int j = 0;j < MAP_SIZE;++j){
            m_mapData[i][j] = '0';
        }
    }

    for(int i = 0;i < MAP_SIZE;++i){
        mvprintw(i + 1 + m_y, m_x - 1, "%c", 'A' + i);
        mvprintw(m_y + m_height, m_x + 2 + i, "%d", 1 + i);
    }

    mvwprintw(m_pWindow, 0, 2, "ATTACKER");
}

Player::~Player(){

}
```

PLAYER.CPP
- DRAW()

- 데이터가 0인 부분은 - 으로, 아닌 부분을 데이터를 출력 한다.

```
void Player::Draw(){
    wattron(m_pWindow, COLOR_PAIR(1));
    for(int i = 0;i < MAP_SIZE;++i){
        for(int j = 0;j < MAP_SIZE;++j){
            mvwprintw(m_pWindow, i + 1, j + 2, "-");
        }
    }
    for(int i = 0;i < MAP_SIZE;++i){
        for(int j = 0;j < MAP_SIZE;++j){
            if (m_mapData[i][j] != '0'){
                mvwprintw(m_pWindow, i + 1, j + 2, "%c", m_mapData[i][j]);
            }
        }
    }
    wattroff(m_pWindow, COLOR_PAIR(1));

    wrefresh(m_pWindow);
}
```

PLAYER.CPP
- UPDATE(), ATTACK()

- update 함수는 지정된 좌표의 데이터를 받아온 데이터로 업데이트한다.

- attack 함수는 지정된 좌표가 이미 공격한 좌표인지 확인하여, 공격한 좌표이면 try-again, 아니면 hit을 리턴한다.

```
void Player::update(int row, int col, char data){
    m_mapData[row][col] = data;
    Draw();
}

std::string Player::attack(int row, int col){
    if (m_mapData[row][col] == 'H' || m_mapData[row][col] == 'M'){
        return "try-again";
    }else{
        return "hit";
    }
}
```

INPUTPANE.H

- Pane을 상속한 InputPane을 정의한다.

```
#include <string>
#include "Pane.h"

class InputPane : public Pane{
    public:
        InputPane(int x, int y, int width, int height);
        ~InputPane();

        void Draw();
        void Draw(char a);
        void Draw(char a, char b, std::string data);
};
```

INPUTPANE.CPP
- 생성자, 소멸자

- < INPUT > 이름을 위에 갖는 객체를 하나 생성한다.

```
InputPane::InputPane(int x, int y, int width, int height)
    :Pane(x, y, width, height)
{
    mvwprintw(m_pWindow, 0, 3, "< INPUT >");
}

InputPane::~InputPane(){

}
```

INPUTPANE.CPP
- DRAW()

- 인자를 받지 않는 Draw는 창을 초기화하는 느낌의 함수이다. 창의 내용을 기본값으로 만든다.

- 인자를 하나 받는 함수는 키보드로부터 입력받은 글자 하나를 inputpane의 윈도우 안에 그린다.

```
void InputPane::Draw(){
    wattron(m_pWindow, COLOR_PAIR(3));
    mvwprintw(m_pWindow, 1, 2, "Input position...(ex: A 3)");
    mvwprintw(m_pWindow, 2, 2, "Input :                    ");
    for(int i = 0;i < 25;i++){
        mvwprintw(m_pWindow, 3, 2 + i, " ");
    }
    wattroff(m_pWindow, COLOR_PAIR(3));
    wrefresh(m_pWindow);
}

void InputPane::Draw(char a){
    wattron(m_pWindow, COLOR_PAIR(3));
    mvwprintw(m_pWindow, 1, 2, "Input position...(ex: A 3)");
    mvwprintw(m_pWindow, 2, 2, "Input : %c", a);
    wattroff(m_pWindow, COLOR_PAIR(3));
    wrefresh(m_pWindow);
}
```

INPUTPANE.CPP - DRAW()

- 인자를 세 개 입력받는 함수는 키보드로부터 입력받은 두개의 글자와 attack의 결과를 출력한다.

```
void InputPane::Draw(char a, char b, std::string data){
    wattron(m_pWindow, COLOR_PAIR(3));
    mvwprintw(m_pWindow, 1, 2, "Input position...(ex: A 3)");
    mvwprintw(m_pWindow, 2, 2, "Input : %c %c", a, b);
    mvwprintw(m_pWindow, 3, 2, "%c", a);
    mvwprintw(m_pWindow, 3, 4, "%c", b);
    for(int i = 0;i < data.length();i++){
        mvwprintw(m_pWindow, 3, 6 + i, "%c", data[i]);
    }
    wattroff(m_pWindow, COLOR_PAIR(3));
    wrefresh(m_pWindow);
}
```

STATPANE.H

- 현재 상태를 나타내는 StatPane을 정의한다.

```h
#include "Pane.h"

class StatPane : public Pane{
    public:
        StatPane(int x, int y, int width, int height);
        ~StatPane();

        void Draw(bool aircraft, bool battleship, bool cruiser, bool destroyer);
        void turnPass();
    protected:
        int turn;
};
```

STATPANE.CPP
- 생성자, 소멸자

- < STATUS > 를 제목으로 하는 윈도우를 하나 만든다. turn은 1로 초기화한다.

```
StatPane::StatPane(int x, int y, int width, int height)
    :Pane(x, y, width, height)
{
    mvwprintw(m_pWindow, 0, 3, "< STATUS >");
    turn = 1;
}

StatPane::~StatPane(){

}
```

STATPANE.CPP - DRAW()

- StatPane을 그릴 때, turn은 일단 출력하고, 배들은 파괴되었다면 빨간색으로, 아니라면 turn과 같은 색으로 출력한다.

- 인자로 받아온 값에 따르고, 기본값은 '파괴되지 않음'이다.

```
void StatPane::Draw(bool aircraft = 0, bool battleship = 0, bool cruiser = 0, bool destroyer = 0){
    wattron(m_pWindow, COLOR_PAIR(2));
    mvwprintw(m_pWindow, 1, 2, "TURN : %d", turn);
    if (aircraft){
        wattroff(m_pWindow, COLOR_PAIR(2));
        wattron(m_pWindow, COLOR_PAIR(8));
        mvwprintw(m_pWindow, 2, 2, "AIRCRAFT : AAAAA");
        wattroff(m_pWindow, COLOR_PAIR(8));
        wattron(m_pWindow, COLOR_PAIR(2));
    }else{
        mvwprintw(m_pWindow, 2, 2, "AIRCRAFT : AAAAA");
    }
```

STATPANE.CPP - TURNPASS()

- turn을 하나 증가시키는 역할을 담당한다.

```
void StatPane::turnPass(){
    turn++;
}
```

SHIP.H
- POSITION

- 배의 위치를 정의하기 위한 position 클래스이다.
- x, y좌표를 갖고 있다.

```
class Position{
    protected:
        int row;
        int col;
    public:
        Position(int _row, int _col);
        ~Position();
        int getRow();
        int getCol();
};
```

SHIP.H
- SHIP

- 배를 정의하는 ship 클래스이다.

- 남은 체력, 배의 이름, 위치, 크기, 방향을 갖고 있다.

```
class Ship{
    protected:
        int remain_HP;
        std::string name;
        Position *position;
        int size;
        bool direction;
    public:
        Ship(int hp, std::string _name, int _row, int _col, int _size, bool _direction);
        ~Ship();
        std::string getName();
        Position getPosition();
        int getRemainHP();
        int getSize();
        bool getDirection();
        void setPosition(int row, int col);
        bool isDestroyed();
        virtual void Draw(WINDOW* win);
        void attacked();
};
```

SHIP.CPP
- POSITION

- Position의 생성자, 소멸자, row를 리턴하는 getRow, col을 리턴하는 getCol을 갖고 있다.

```cpp
Position::Position(int _row, int _col){
    row = _row;
    col = _col;
}

Position::~Position(){

}

int Position::getRow(){
    return row;
}

int Position::getCol(){
    return col;
}
```

SHIP.CPP
- SHIP - 생성자, 소멸자

- 인자로 받아온 값들을 대입 해준다.

```
Ship::Ship(int hp, std::string _name, int _row, int _col, int _size, bool _direction){
    remain_HP = _size;
    name = _name;
    position = new Position(_row, _col);
    size = _size;
    direction = _direction;
}

Ship::~Ship(){

}
```

SHIP.CPP
- SHIP - GETTER

- 각 값들을 리턴하는 getter 함수이다.

```
std::string Ship::getName(){
    return name;
}

Position Ship::getPosition(){
    return *position;
}

int Ship::getRemainHP(){
    return remain_HP;
}

int Ship::getSize(){
    return size;
}

bool Ship::getDirection(){
    return direction;
}
```

SHIP.CPP
- SHIP - SETPOSITION()

- 배에 새로운 위치를 대입하는 함수이다.

```
void Ship::setPosition(int _row, int _col){
    delete position;
    position = new Position(_row, _col);
}
```

SHIP.CPP
- SHIP - ISDESTROYED()

- 남은 hp가 0과 같은지 확인
  해서 파괴되었는지 확인한다.

```
bool Ship::isDestroyed(){
    return remain_HP == 0;
}
```

SHIP.CPP
- SHIP - ATTACKED()

- 공격받았다면, hp를 1 감소시킨다.

```
void Ship::attacked(){
    remain_HP -= 1;
}
```

AIRCRAFT.H

- Aircraft를 정의하기 위해서 사용한다.

```h
#include "Ship.h"

class Aircraft : public Ship{
    public:
        Aircraft(int _row, int _col, bool _direction);
        ~Aircraft();
        void Draw(WINDOW* win);
};
```

AIRCRAFT.CPP
- 생성자, 소멸자

- Aircraft는 5칸이므로, hp는 5, 이름은 Aircraft, row는 _row로, col은 _col으로, size는 5로, direction은 _direction으로 초기화한다.

```
Aircraft::Aircraft(int _row, int _col, bool _direction)
    :Ship(5, "Aircraft", _row, _col, 5, _direction)
{

}

Aircraft::~Aircraft(){

}
```

AIRCRAFT.CPP
- DRAW()

- window를 인자로 받아서 그 윈도우 위에 배를 그린다.

- 방향을 구분하여 적절히 배를 배치한다.

```
void Aircraft::Draw(WINDOW *win){
    Position p = this -> getPosition();
    int row = p.getRow();
    int col = p.getCol();
    bool direction = this -> getDirection();
    wattron(win, COLOR_PAIR(4));
    if (!direction){
        for(int i = 0;i < this -> getSize();++i){
            mvwprintw(win, i + row + 1, col + 2, "%c", this -> name[0]);
        }
    }else{
        for(int i = 0;i < this -> getSize();++i){
            mvwprintw(win, row + 1, i + col + 2, "%c", this -> name[0]);
        }
    }
    wattroff(win, COLOR_PAIR(4));
    wrefresh(win);
}
```

- 다른 배들에 대해서도 기본적인 로직은 같다.
- 다만 배들을 색깔로 구분해주기 위해서 DRAW 함수 부분에 차이고 있고, 배를 선언할 때 size와 hp에 들어가는 숫자가 다르다.