## C++ 프로그래밍 및 실습

# 4인용 십이장기 구현

### 진척 보고서 #1

제출일자 : 2024년 11월 17일

제출자명 : 이재혁

제출자학번 : 213161

# 1. 프로젝트 목표 (16 pt)

## 1) 배경 및 필요성 (14 pt)

최근 예전 TV 프로그램인 '더 지니어스'에 빠지게 되었고, 이 프로그램의 게임 중 하나를 C++ 프로그램으로 구현해 보면 어떨까 하는 생각이 들었습니다. '더 지니어스'는 복잡한 심리전과 전략이 중요한 게임들이 많아, 여러 명이 참여하여 역동적인 전략을 펼칠 수 있는 십이장기를 다인용으로 확장하면 재미있을 것 같았습니다. 기존 십이장기는 두 명만 플레이할 수 있는 전통 보드게임으로, 제한된 공간과 기물로 인해 전략적 깊이에 한계가 있습니다. 이에 따라 4명의 플레이어가 동시에 참여할 수 있도록 십이장기 규칙을 변형하고, 5x5 보드를 통해 더 넓은 공간에서 다양한 전략을 펼칠 수 있는 게임을 구현하고자 합니다.

## 2) 프로젝트 목표

이 프로젝트의 목표는 십이장기를 다인용 게임으로 확장하여 4명의 플레이어가 각자 자신의 기물을 움직이며 상대방을 공격하고 방어할 수 있는 게임을 C++로 구현하는 것입니다. 상대방의 기물을 잡으면 자신의 기물로 사용할 수 있어 더욱 다양한 전략을 펼칠 수 있으며, 이를 통해 플레이어 간 상호작용과 역전의 묘미를 강화하고자 합니다.

## 3) 차별점

기존 십이장기는 두 명의 플레이어가 고정된 3x4 보드에서 제한된 기물로 대결하는 방식입니다. 본 프로젝트에서는 4명이 플레이할 수 있도록 보드를 5x5로 확장하고, 잡은 기물을 자신의 기물로 사용할 수 있는 독창적인 규칙을 추가하여 게임의 전략성을 높였습니다.

## 2. 기능 계획

### 1) 기능 1 5x5 보드 초기화

- 설명 : 4명의 플레이어가 사용할 수 있도록 5x5 크기의 보드를 초기화하고, 각 플레이어의 기물을 배치합니다.

**(1) 세부 기능 1: 기물 배치**

- 각 플레이어는 왕(KING), 나이트(KNIGHT), 병사(SOLDIER) 기물을 각자의 시작 위치에 배치하며, 이를 4명의 플레이어에 맞춰 각 모서리에 배치합니다.

**(2) 세부 기능 2: 중앙 영역 설정**

- 보드의 중앙 영역은 주요 전투 구역으로, 각 플레이어가 중앙을 차지하기 위해 경쟁할 수 있도록 배치합니다.

### 2) 기능 2 보드 출력

- 설명: 보드의 현재 상태를 출력하여 각 플레이어가 자신의 기물 위치와 상대방의 기물 위치를 확인할 수 있도록 합니다.

(1) 세부 기능 1 : 기물 및 플레이어 표시

- 기물 종류(KING, KNIGHT, SOLDIER)와 소유한 플레이어 번호(PLAYER1, PLAYER2 등)를 함께 표시하여 현재 게임 상황을 쉽게 파악할 수 있게 합니다.

# 3. 진척사항

## 1) 기능 구현

### (1) 구현한 기능 이름 : 보드 초기화 및 출력

- 입력: 없음 (초기 상태만 출력)

  - 출력: 5x5 보드에서 각 플레이어의 기물이 배치된 상태를 출력

- 적용된 프로그래밍 내용:

  - 열거형(enum)을 사용하여 기물 종류와 소유자를 정의.

```
// 기물 종류를 나타내는 열거형
enum PieceType { EMPTY, KING, KNIGHT, SOLDIER };

// 플레이어를 나타내는 열거형
enum Player { NONE, PLAYER1, PLAYER2, PLAYER3, PLAYER4 };
```

  - 구조체(struct)를 사용하여 기물 데이터를 관리.

```
// 기물 구조체 정의
struct Piece {
    PieceType type;
    Player owner;
};
```

  - 2차원 벡터(vector)를 사용하여 보드 상태를 동적으로 관리.

```
// 보드를 초기화하는 함수
vector<vector<Piece>> initializeBoard() {
    vector<vector<Piece>> board(BOARD_SIZE, vector<Piece>(BOARD_SIZE, { EMPTY, NONE }));

    // PLAYER1의 기물 배치
    board[0][0] = { KING, PLAYER1 };
    board[0][1] = { KNIGHT, PLAYER1 };
    board[1][0] = { SOLDIER, PLAYER1 };

    // PLAYER2의 기물 배치
    board[0][4] = { KING, PLAYER2 };
    board[0][3] = { KNIGHT, PLAYER2 };
    board[1][4] = { SOLDIER, PLAYER2 };

    // PLAYER3의 기물 배치
    board[4][4] = { KING, PLAYER3 };
    board[4][3] = { KNIGHT, PLAYER3 };
    board[3][4] = { SOLDIER, PLAYER3 };

    // PLAYER4의 기물 배치
    board[4][0] = { KING, PLAYER4 };
    board[4][1] = { KNIGHT, PLAYER4 };
    board[3][0] = { SOLDIER, PLAYER4 };

    return board;
}
```

- 조건문(if-else)을 사용하여 출력 논리를 구현.

```
if (piece.owner == NONE) {
    cout << "  빈칸  |";
} else {
    cout << " ";
    switch (piece.owner) {
        case PLAYER1: cout << "P1_"; break;
        case PLAYER2: cout << "P2_"; break;
        case PLAYER3: cout << "P3_"; break;
        case PLAYER4: cout << "P4_"; break;
        default: break;
    }
    switch (piece.type) {
        case KING: cout << "왕 "; break;
        case KNIGHT: cout << "기사"; break;
        case SOLDIER: cout << "병사"; break;
        default: break;
    }
    cout << " |";
}
```

- 테스트 결과 :

초기 보드 상태가 예상대로 출력되었으며, 각 플레이어의 기물이 정확히 배치됨을 확인.

출력 예시 :

```
초기 5x5 십이장기 보드 상태:
=========================================
            십이장기 보드 상태
=========================================
| P1_왕  | P1_기사 |  빈칸  | P2_기사 | P2_왕  |
-----------------------------------------
| P1_병사 |  빈칸  |  빈칸  |  빈칸  | P2_병사 |
-----------------------------------------
|  빈칸  |  빈칸  |  빈칸  |  빈칸  |  빈칸  |
-----------------------------------------
| P4_병사 |  빈칸  |  빈칸  |  빈칸  | P3_병사 |
-----------------------------------------
| P4_왕  | P4_기사 |  빈칸  | P3_기사 | P3_왕  |
-----------------------------------------
```

# 4. 계획 대비 변경 사항

## 1) 변경 내역 제목

1) 변경 내역 제목: 보드 크기 조정

- 이전: 4x4 보드
- 이후: 5x5 보드
- 사유: 플레이어 간 상호작용을 강화하고, 이동 및 전략 공간을 확보하기 위해.

## 2) 변경 예정 내용

1) 변경 내역 제목 : 게임의 룰 변경

- 미정

2) 변경 내역 제목 : 게임의 말 변경

- 미정

# 5. 프로젝트 일정