프로젝트 실습(tic_tac_toe) 보고서

소프트웨어공학과
190025 안호균

1. 서론
    (1) 프로젝트 목적 및 배경: 4주차까지 배운 내용에 대한 실습을 위해 진행
    (2) 목표: Tic Tac Toe 게임 구현
2. 요구사항
    (1) 사용자 요구사항: 두 명의 사용자가 번갈아가며 O와 X를 놓기
    (2) 기능 요구사항:
        ① 누구의 차례인지 출력
        ② 좌표 입력 받기
        ③ 입력 받은 좌표 유효성 체크
        ④ 좌표에 O / X 놓기
        ⑤ 현재 보드판 출력
        ⑥ 빙고 시 승자 출력 후 종료
        ⑦ 모든 칸이 찼으면 종료
3. 설계 및 구현
    (1) 기능 별 구현 사항

● 기능 구현에 쓰일 변수 및 배열 선언 및 초기화

```
const int numCell = 3;
int count = 0; // 돌을 둔 횟수를 체크하는 변수 count
char board[numCell][numCell]{};
int x, y; // 사용자에게 입력받는 x, y 좌표를 저장할 변수
```

● 게임 진행 상황을 출력할 보드판을 공백으로 초기화

```
// 보드판 초기화
for (x = 0; x < numCell; x++) {
    for (y = 0; y < numCell; y++) {
        board[x][y] = ' ';
    }
}
```

● 게임 진행에 쓰일 변수 선언 및 초기화

```
// 게임하는 코드
int k = 0; // 누구 차례인지 체크하기 위한 변수
char currentUser = 'X'; // 현재 유저의 볼을 저장하기 위한 문자 변수
```

① 누구의 차례인지 출력

```
while (true) {
// 1. 누구 차례인지 출력
switch (k % 2) {
case 0:
    cout << k % 2 + 1 << "번 유저(X)의 차례입니다 -> ";
    currentUser = 'X';
    break;
case 1:
    cout << k % 2 + 1 << "번 유저(O)의 차례입니다 -> ";
    currentUser = 'O';
    break;
}
```

▶ 입력

1. k = 차례를 체크하는 변수
2. currentUser = 현재 유저의 심볼

▶ 결과

1. k번 유저의 차례임을 출력
2. 현재 유저의 심볼 출력
3. 출력 이후 switch-case문 탈출

▶ 설명: 차례를 체크하기 위한 변수 k를 2로 나눈 나머지를 통해 1번 유저와 2번 유저의 차례를 구분하여 출력한다.

② 좌표 입력 받기

```
// 2. 좌표 입력 받기
cout << "(x, y) 좌표를 입력하세요: ";
cin >> x >> y;
count++;
```

▶ 입력

1. x = 좌표 x 값
2. y = 좌표 y 값
3. count = 유저가 돌을 둔 횟수

▶ 결과

1. 좌표(x, y)를 입력 받음
2. 입력 받은 후 유저가 돌을 둔 횟수 1 증가

▶ 설명: 유저가 돌을 둘 좌표(x, y)를 입력 받고, 돌을 둔 횟수를 1 증가시킨다.

③ 입력 받은 좌표 유효성 체크

```
// 3. 입력받은 좌표의 유효성 체크
if (x >= numCell || y >= numCell) {
    cout << x << ", " << y << ": ";
    cout << " x 와 y 둘 중 하나가 칸을 벗어납니다." << endl;
    continue;
}
if (board[x][y] != ' ') {
    cout << x << ", " << y << ": 이미 돌이 차있습니다." << endl;
    continue;
}
```

▶ 입력

1. x = 좌표 x 값
2. y = 좌표 y 값
3. numCell = 가로/세로 칸의 개수

▶ 결과

1. 칸을 놓을 수 없는 이유를 출력
2. 출력 후 while 문 초반으로 이동

▶ 설명: 유저가 입력한 좌표가 게임 판을 벗어나는지, 입력한 좌표에 이미 돌이 있는지를 체크한다.

④ 좌표에 O / X 놓기

```
// 4. 입력받은 좌표에 현재 유저의 돌 뽑기
board[x][y] = currentUser;
```

▶ 입력

1. x = 좌표 x 값
2. y = 좌표 y 값
3. currentUser = 현재 유저의 심볼

▶ 결과

1. 좌표(x, y)에 현재 유저의 심볼(돌)을 놓는다.

▶ 설명: 유저가 입력한 좌표에 해당하는 2차원 배열 board의 인덱스에 돌을 놓는다.

⑤ **현재 보드판 출력**

```
// 5. 현재 보드 판 출력
for (int i = 0; i < numCell; i++) {
    cout << "---|---|---" << endl;
    for (int j = 0; j < numCell; j++) {
        cout << board[i][j];
        if (j == numCell - 1) {
            break;
        }
        cout << "  |";
    }
    cout << endl;
}
cout << "---|---|---" << endl;
k++;
```

▶ 입력

1. i = 반복문의 인덱스
2. j = 중첩 반복문의 인덱스
3. numCell = 가로/세로 칸의 개수
4. k = 차례를 체크하는 변수

▶ 결과

1. numCell의 크기에 맞게 보드 판을 출력한다.
2. 차례를 체크하기 위한 변수 k를 1만큼 증가시킨다.

▶ 설명: 중첩 for문을 활용해 2차원 배열 board에 쓰일 인덱스 i와 j를 생성하고, numCell의 크기만큼 보드 판 모양을 만들어줄 문자열을 출력한다. 이후, 차례를 체크하기 위해 k를 1만큼 증가시킨다.

⑥ 빙고 시 승자 출력 후 종료(가로, 세로줄 확인)

```
// 6. 빙고 시 승자 출력 후 종료
for (int i = 0; i < numCell; i++) {
    // 6-1. 가로 줄의 돌이 모두 같다면
    if ((board[i][0] == board[i][1]) && (board[i][1] == board[i][2]) && board[i][0] != ' ') {
        if (board[i][0] == 'X') {
            cout << "가로에 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!" << endl; // 가로에 모두 놓인 돌이 X라면 X의 승리 출력
            cout << "종료합니다" << endl;
            return 0;
        }
        else {
            cout << "가로에 모두 돌이 놓였습니다!: 2번 유저(O)의 승리입니다!" << endl; // 가로에 모두 놓인 돌이 O라면 O의 승리 출력
            cout << "종료합니다" << endl;
            return 0;
        }
    }
    for (int j = 0; j < numCell; j++) {
        // 6-2. 세로 줄의 돌이 모두 같다면
        if ((board[0][j] == board[1][j]) && (board[1][j] == board[2][j]) && board[0][j] != ' ') {
            if (board[0][j] == 'X') {
                cout << "세로에 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!" << endl; // 세로에 모두 놓인 돌이 X라면 X의 승리 출력
                cout << "종료합니다" << endl;
                return 0;
            }
            else {
                cout << "세로에 모두 돌이 놓였습니다!: 2번 유저(O)의 승리입니다!" << endl; // 세로에 모두 놓인 돌이 O라면 O의 승리 출력
                cout << "종료합니다" << endl;
                return 0;
            }
        }
    }
}
```

▶ 입력

1. i = 반복문의 인덱스
2. j = 중첩 반복문의 인덱스
3. numCell = 가로/세로 칸의 개수

▶ 결과

1. 가로 줄에 놓인 돌이 모두 같다면 해당 유저(O/X)의 승리를 출력한다.
2. 세로 줄에 놓인 돌이 모두 같다면 해당 유저(O/X)의 승리를 출력한다.
3. 승리 조건이 만족되었다면 프로그램을 종료한다.

▶ 설명: 중첩 for문을 활용해 2차원 배열 board에 쓰일 인덱스 i와 j를 생성하고, 가로 줄에 놓인 돌을 비교하기 위해 board[i][0]에 놓인 돌이 모두 같은지를 비교한다. 만약 board[i][0~2]에 놓인 돌이 모두 같다면 해당 심볼이 무엇인지를 확인하고, 해당 심볼을 가진 유저의 승리를 출력한다. 세로 줄도 같은 방식으로 board[0~2][j]에 놓인 돌이 모두 같다면 해당 심볼이 무엇인지를 확인하고, 해당 심볼을 가진 유저의 승리를 출력한다.

⑥ 빙고 시 승자 출력 후 종료(2가지 대각선 확인)

```
// 6-3. 대각선의 돌이 모두 같다면
// 왼쪽 위에서 오른쪽 아래 대각선의 돌이 모두 같은 경우
if ((board[0][0] == board[1][1] && board[1][1] == board[2][2]) && board[1][1] != ' ') {
    if (board[0][0] == board[1][1] && board[1][1] == 'X') { // 위 조건을 만족하면서 놓인 돌이 X일 경우
        cout << "왼쪽 위에서 오른쪽 아래 대각선으로 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!" << endl; // X의 승리 출력
        cout << "종료합니다" << endl;
        return 0;
    }
    else if(board[0][0] == board[1][1] && board[1][1] == '0') { // // 위 조건을 만족하면서 놓인 돌이 0일 경우(특정 좌표 1개만을 비교하므로 단순히 else문이 아닌 else if를 사용함)
        cout << "왼쪽 위에서 오른쪽 아래 대각선으로 모두 돌이 놓였습니다!: 2번 유저(0)의 승리입니다!" << endl; // 0의 승리 출력
        cout << "종료합니다" << endl;
        return 0;
    }
}
// 오른쪽 위에서 왼쪽 아래 대각선의 돌이 모두 같은 경우
if ((board[0][2] == board[1][1] && board[1][1] == board[2][0]) && board[1][1] != ' ') {
    if (board[0][2] == board[1][1] && board[1][1] == 'X') { // 위 조건을 만족하면서 놓인 돌이 X일 경우
        cout << "오른쪽 위에서 왼쪽 아래 대각선으로 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!" << endl; // X의 승리 출력
        cout << "종료합니다" << endl;
        return 0;
    }
    else if (board[0][2] == board[1][1] && board[1][1] == '0') { // // 위 조건을 만족하면서 놓인 돌이 0일 경우(특정 좌표 1개만을 비교하므로 단순히 else문이 아닌 else if를 사용함)
        cout << "오른쪽 위에서 왼쪽 아래 대각선으로 모두 돌이 놓였습니다!: 2번 유저(0)의 승리입니다!" << endl; //  0의 승리 출력
        cout << "종료합니다" << endl;
        return 0;
    }
}
```

▶ 입력

1. 위 코드에서 채워진 2차원 배열 board[][]

▶ 결과

1. 왼쪽 위에서 오른쪽 아래 대각선으로 놓인 돌이 모두 같다면 해당 유저(O/X)의 승리를 출력한다.
2. 오른쪽 위에서 왼쪽 아래 대각선으로 놓인 돌이 모두 같다면 해당 유저(O/X)의 승리를 출력한다.
3. 승리 조건이 만족되었다면 프로그램을 종료한다.

▶ 설명: 왼쪽 위에서 오른쪽 아래 대각선으로 놓인 돌을 비교하기 위해 board[0][0], board[1][1], board[2][2]에 놓인 돌이 모두 같은지를 비교한다. 만약 board[0][0], board[1][1], board[2][2]에 놓인 돌이 모두 같다면 해당 심볼이 무엇인지를 확인하고, 해당 심볼을 가진 유저의 승리를 출력하고 프로그램을 종료한다. 오른쪽 위에서 왼쪽 아래 대각선으로 놓인 돌도 같은 방식으로 board[0][2], board[1][1], board[2][0]에 놓인 돌이 모두 같다면 해당 심볼이 무엇인지를 확인하고, 해당 심볼을 가진 유저의 승리를 출력하고 프로그램을 종료한다.

⑦ 모든 칸이 찼으면 종료

```
// 7. 모든 칸이 찼으면 종료
if (count == 9) { // 돌을 둔 횟수가 9회이면서, 위의 모든 승리 조건이 만족되지 않은 경우에만 실행
    cout << "모든 칸이 다 찼습니다. 종료합니다." << endl;
    return 0;
}
```

▶ 입력

1. count = 돌을 둔 횟수

▶ 결과

1. 돌을 둔 횟수가 9회(더 이상 둘 곳이 없는 경우)이면서, 유저의 모든 승리 조건이 만족되지 않은 경우, 모든 칸이 찼음을 안내하고 종료한다.

▶ 설명: 앞서 구현한 모든 승리 조건이 만족되지 않고 마지막 이 조건문까지 내려왔다면, 더 이상 유저가 돌을 둘 빈칸이 없다는 의미이다. 따라서 count 변수의 값이 9가 되면 모든 칸이 찼음을 안내하고 프로그램을 종료한다.

4. 테스트

(1) 기능별 테스트 결과

① 누구의 차례인지 출력

```
1번 유저(X)의 차례입니다
```

② 좌표 입력 받기

```
(x, y) 좌표를 입력하세요: 0 0
```

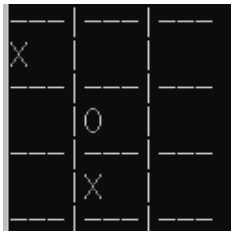
③ 입력 받은 좌표 유효성 체크

```
2번 유저(O)의 차례입니다 -> (x, y) 좌표를 입력하세요: 3 3
3, 3:  x 와 y 둘 중 하나가 칸을 벗어납니다.
2번 유저(O)의 차례입니다 -> (x, y) 좌표를 입력하세요: 0 0
0, 0: 이미 돌이 차있습니다.
```

④ 좌표에 O / X 놓기

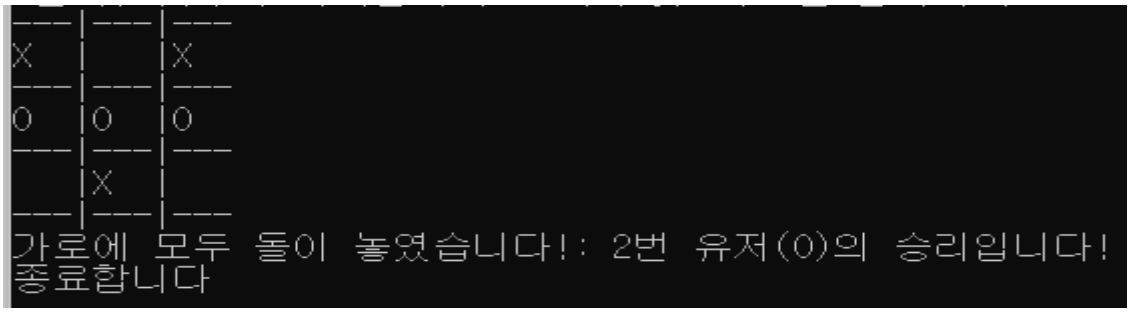
```
1번 유저(X)의 차례입니다 -> (x, y) 좌표를 입력하세요: 0 0
---|---|---
X  |   |
---|---|---
   |   |
---|---|---
   |   |
---|---|---
2번 유저(O)의 차례입니다 -> (x, y) 좌표를 입력하세요: 1 1
---|---|---
X  |   |
---|---|---
   | O |
---|---|---
   |   |
---|---|---
```

⑤ 현재 보드판 출력

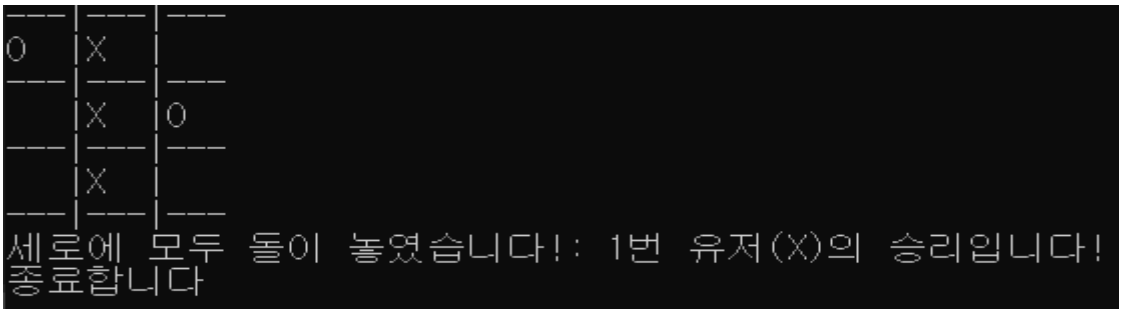


⑥ 빙고 시 승자 출력 후 종료

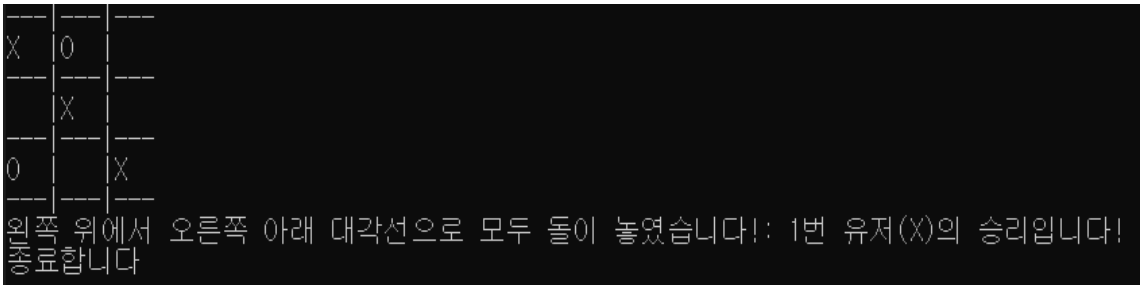
<가로로 놓인 돌이 모두 같은 경우>



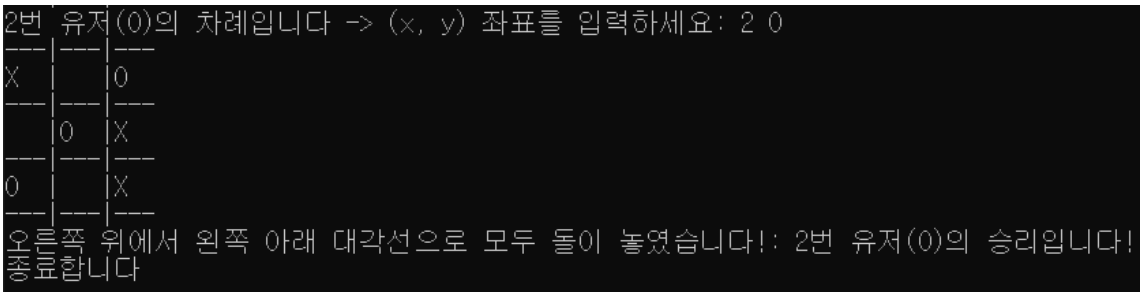
<세로로 놓인 돌이 모두 같은 경우>



<왼쪽 위에서 오른쪽 아래 대각선으로 놓인 돌이 모두 같은 경우>



<오른쪽 위에서 왼쪽 아래 대각선으로 놓인 돌이 모두 같은 경우>



⑦ 모든 칸이 찼으면 종료

5. 결과 및 결론

1. 프로젝트 결과: Tic Tac Toe 게임을 제작했음.

2. 느낀 점: 처음부터 중간고사 범위까지 배운 모든 내용을 응용하며 배운 것을 적용해보았다. 알고리즘을 직접 구현해보면서 다양한 자료형과 변수를 다루며 입력을 받고, 그에 따른 출력을 해보기도 했으며, 이를 활용하기 위해 각종 연산자를 사용하여 조건문과 반복문을 적절히 다루었다. 또한, 이 과정에서 걸러진 값을 2차원 배열에 저장하였고, 조건문과 반복문을 다시 한번 활용하여 올바른 결과물을 도출하며 모든 요구사항을 충족할 수 있었다. 지금까지 배운 내용을 몸으로 익힐 수 있었던 좋은 기회였다.