Tic Tac Toe 게임

1.서론

1-1. 프로젝트 목적 및 배경

· 4주차까지 배운 내용에 대한 실습을 위해 진행

1-2. 목표

· Tic Tac Toe 게임 구현

2.요구사항

2-1. 사용자 요구사항

· 두명의 사용자가 번갈아가며 O와 X를 놓기

2-2. 기능 요구사항

· 누구의 차례인지 출력
· 좌표 입력 받기
· 입력 받은 좌표 유효성 체크
· 좌표에 O/X 놓기
· 현재 보드판 출력
· 빙고 시 승자 출력 후 종료
· 모든 칸이 찼으면 종료

3.설계 및 구현

1. 누구 차례인지 출력

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

1. 입력
   · k = (반복 횟수 - 1)
   · currentUser = 현재 유저의 돌의 모양 변수
2. 결과
   · 현재 차례를 출력
   · 현재 차례의 유저의 돌의 모양 대입
3. 설명
   · 반복횟수의 2로 나눈 나머지를 이용하여 차례 출력
   · 현재 차례의 돌의 모양(문자)으로 대입

2. 좌표 입력 받기

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

1. 입력
   · x = x좌표값
   · y = y좌표값
2. 결과
   · 돌을 놓을 위치 사용자 입력
3. 설명
   · cin을 이용하여 x, y의 값을 사용자 입력 및 대입

3. 입력받은 좌표의 유효성 체크

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

1. 입력
   · x = x좌표값
   · y = y좌표값
   · numCell = 가로/세로 칸 개수
   · board[][] = 빙고의 칸과 각각 대응되는 배열
2. 결과
   · 칸을 놓을 수 없는 이유를 출력
   · 출력 후 while문 초반으로 이동
3. 설명
   · 사용자가 입력한 좌표가 게임판을 벗어나는지 if로 체크
   · 사용자가 입력한 좌표에 돌이 이미 있는지 if로 체크

4. 좌표에 O/X 놓기

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

1. 입력
   · x = x좌표값
   · y = y좌표값
   · currentUser = 현재 유저의 돌의 모양 변수
   · board[][] = 빙고의 칸과 각각 대응되는 배열
2. 결과
   · 빙고의 칸에 돌을 넣기
3. 설명
   · 빙고의 칸에 대응되는 배열에 x, y에 해당하는 위치에 돌의 모양 대입

5. 현재 보드판 출력

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

1. 입력
   · numCell = 가로/세로 칸 개수
   · board[][] = 빙고의 칸과 각각 대응되는 배열
2. 결과
   · 현재의 보드 판 출력
3. 설명
   · 첫 번째 for문은 y좌표 2부터 내림차순으로 변경
   · 두 번째 for문은 x좌표 0부터 오름차순으로 변경
   · 위에 따라 순서대로 빙고판에 들어있는 값과 경계선 출력

6. 빙고 시 승자 출력 후 종료 / 모든 칸이 찼으면 종료

```
//6. 빙고 시 승자 출력 후 종료(현재 놓은 돌 위치 기준 비교)
if(k >= 4){     //빙고는 5개의 돌이 놓인 후부터 가능
    //6-1. 가로
    if(board[y][0] == board[y][1] && board[y][0] == board[y][2]){
        switch(k % 2){
            case 0:
                cout << "첫번째 유저 빙고!";
                return 0;
            case 1:
                cout << "두번째 유저 빙고!";
                return 0;
        }
    }

    //6-2. 세로
    if(board[0][x] == board[1][x] && board[0][x] == board[2][x]){
        switch(k % 2){
            case 0:
                cout << "첫번째 유저 빙고!";
                return 0;
            case 1:
                cout << "두번째 유저 빙고!";
                return 0;
        }
    }
```

```
//6-3. 대각선
if((x + y) % 2 == 0) {
    if(board[1][1] != ' ' && board[0][0] == board[1][1] && board[1][1] == board[2][2]){
        switch(k % 2){
            case 0:
                cout << "첫번째 유저 빙고!";
                return 0;
            case 1:
                cout << "두번째 유저 빙고!";
                return 0;
        }
    }
    if(board[1][1] != ' ' && board[0][2] == board[1][1] && board[1][1] == board[2][0]){
        switch(k % 2){
            case 0:
                cout << "첫번째 유저 빙고!";
                return 0;
            case 1:
                cout << "두번째 유저 빙고!";
                return 0;
        }
    }
}

//6-4. 모든 칸이 찼을 경우
if(k == 8){
    cout << "모든 칸이 찼습니다.";
    return 0;
}
```

1. 입력
 · x = x좌표값
 · y = y좌표값
 · k = (반복 횟수 - 1)
 · board[][] = 빙고의 칸과 각각 대응되는 배열

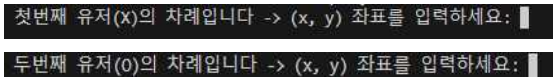
2. 결과
 · 빙고 시 승자 출력 후 종료
 · 빙고 칸이 다 찼을 시 종료

3. 설명
 //공통
 · 빙고는 5개의 돌이 놓인 후부터 가능하기에 if로 체크
 · 모든 switch문은 k를 2로 나눈 나머지값으로 현재 차례의 사람의 승리 출력
 · 직전에 놓인 돌이 승리와 관련된 위치임으로 그 돌을 기준
 //6-1
 · 직전에 놓인 돌의 가로줄이 다 같은지 if문으로 체크
 //6-2
 · 직전에 놓인 돌의 세로줄이 다 같은지 if문으로 체크
 //6-3
 · 직전에 놓인 돌의 x, y의 합이 짝수일 경우에만 대각선이 가능하기에 if문으로 체크
 · 2개의 대각선 중 공백이 아닌 같은 돌로 채워진 것이 있는지 if문으로 체크
 //6-4
 · 빙고판이 가득 찼을 경우를 if문으로 체크

4. 테스트

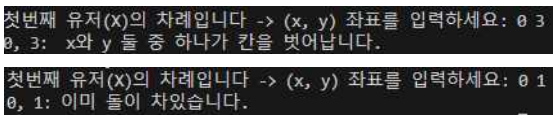
4-1. 기능 별 테스트 결과

1. 누구 차례인지 출력



2. 좌표 입력 받기

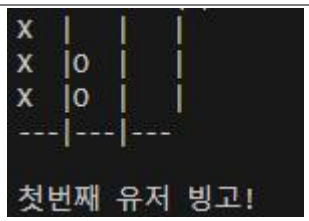
3. 입력받은 좌표의 유효성 체크

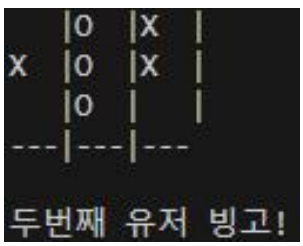


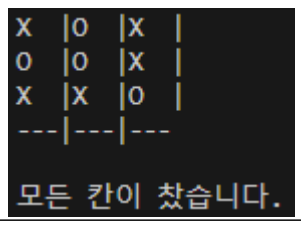
4. 좌표에 O/X 놓기

5. 현재 보드판 출력

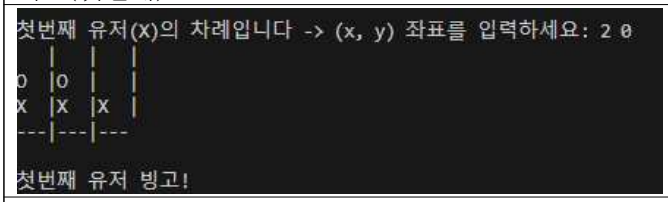
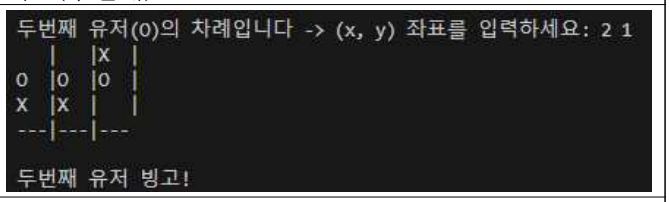
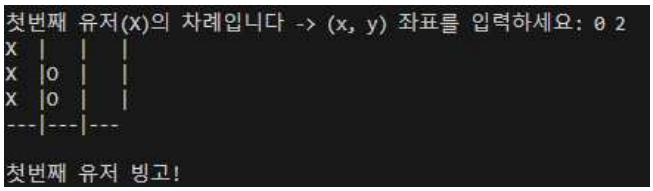
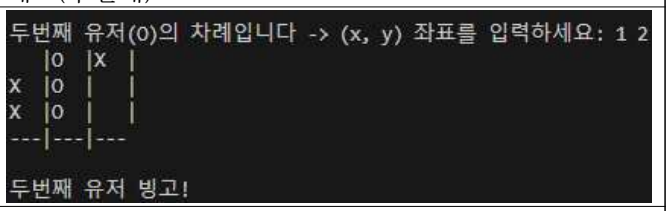
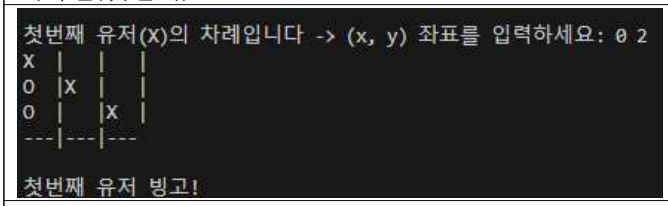
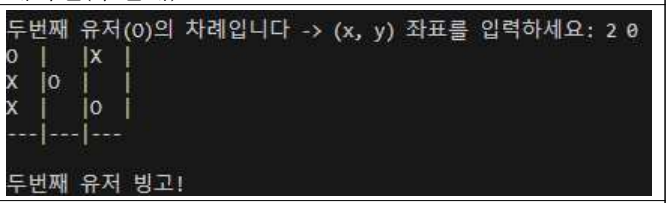
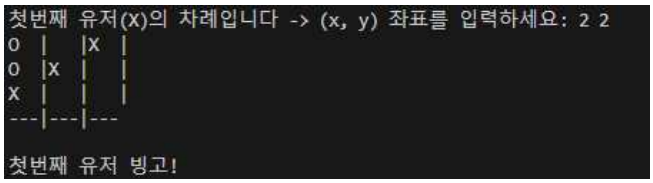
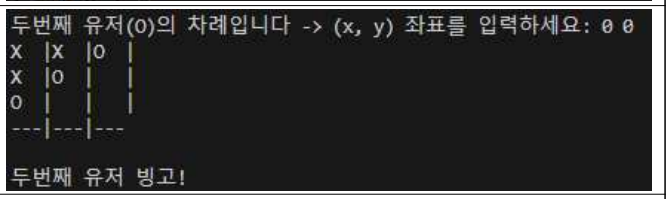
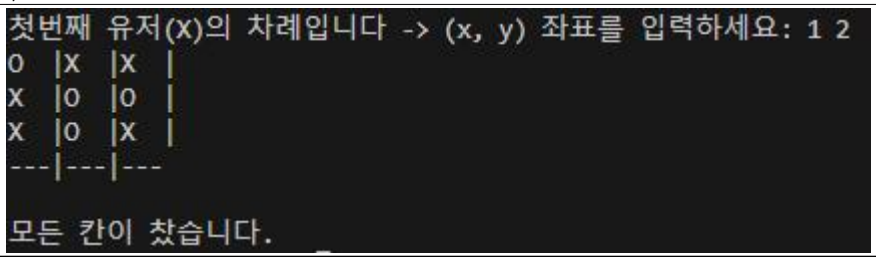
6. 빙고 시 승자 출력 후 종료 / 모든 칸이 찼으면 종료







4-2.최종 테스트 스크린샷

<table>
<tr><td>가로(첫번째)</td><td>가로(두번째)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>세로(첫번째)</td><td>세로(두번째)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>대각선(첫번째)</td><td>대각선(두번째)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">모든 칸이 찼을 경우</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

5. 결과 및 결론

5-1. 프로젝트 결과

· 2차 배열, 반복문(for문, while문), 조건문(if문, switch문), break문, continue문 cin, cout, 논리연산자을 활용하여 Tic Tac Toe 게임을 만들었음

5-2. 느낀 점

· 코드가 조금 길어지다보니 지저분한 느낌이 들었습니다. 그리고 반복되어 사용되는 기능이 있다보니 중복 코드가 많아 중복코드 중 잘못된 부분이 있다면 모두 찾아가 수정해야 하는 경우가 번거로웠습니다. 그래서 주석을 적극적으로 활용하고 중복 코드는 그 기능의 함수를 만들어 호출해서 사용해야 겠다는 생각이 들었습니다. 또한 코드를 이해하기 쉽고 깔끔하게 하기 위해서는 많은 코드를 접하고 경험이 필요하다는 생각이 들었습니다.

· 특정 기능을 구현하는 것은 어떻게든 할 수 있지만 메모리 효율, 시간 효율을 생각하면서 작성하는 것은 쉽지 않다는 생각이 들었습니다. 그렇기에 많은 코드와 알고리즘을 접하고 끊임없이 고민해봐야 하며 많은 경험을 해봐야 겠다는 생각이 들었습니다.