Tic Tac Toe 게임

182671 김준석

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 :

(1) 누구의 차례인지 출력

1. 코드블록 스크린샷

```
// 게임을 진행하는 코드
// 현재 차례인 사용자를 나타내는 int형 변수(0또는 짝수이면 X, 홀수이면 O)
int k = 0;
char currentUser = 'X'; // 현재 사용자의 돌을 저장할 변수
while (true) {
    // 1. 현재 차례인 사용자를 출력한다.
    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;
    }
```

2. 입력

    - k => 현재 차례인 사용자를 나타내는 변수

    - currentUser => 현재 사용자의 돌을 저장하는 문자열 변수

3. 결과

    - k % 2의 결과로 case문 수행

    - k를 갖고 연산하여 결과값 출력

    - case에 따라 currentUser에 문자열 저장

4. 설명

    - k를 2로 나눈 나머지 값으로 현재 사용자를 파악한다.

    - 나머지가 0 또는 짝수 -> X / 홀수 -> O

    - 이렇게 파악한 사용자의 돌을 currentUser에 저장한다.

(2) 좌표 입력 받기

1. 코드블록 스크린샷

```
    // 2. 사용자에게 좌표를 입력받는다.
    cout << "(x, y) 좌표를 입력하세요: ";
    cin >> x >> y;
```

2. 입력

    - 없음.

3. 결과

- x, y좌표를 입력하라는 문자열을 출력

- 사용자로부터 입력받은 문자열을 변수 x, y에 저장

4. 설명

- 현재 차례의 사용자로부터 돌을 놓을 x, y좌표를 입력받는다.

(3) 입력 받은 좌표 유효성 체크

1. 코드블록 스크린샷

```
// 3. 입력한 좌표가 유효한지 체크한다.
// 입력 받은 x, y 좌표 중 보드판을 벗어나는 case
if (x >= numCell || y >= numCell) {
    cout << x << ", " << y << ": ";
    cout << " x 와 y 둘 중 하나가 칸을 벗어납니다." << endl;
    continue;
}
// 입력 받은 좌표 자리에 이미 돌이 있는 case
if (board[x][y] != ' ') {
    cout << x << ", " << y << ": 이미 돌이 차있습니다." << endl;
    continue;
}
```

2. 입력

- x => 사용자로부터 입력받은 x좌표

- y => 사용자로부터 입력받은 y좌표

- numCell => 보드판의 가로 및 세로 칸 개수

3. 결과

- 해당 좌표의 칸에 돌을 놓을 수 없는 이유를 출력

- 문자열을 출력한 후 다시 while-loop 처음으로 이동

4. 설명

- 사용자가 입력한 좌표가 보드 판의 크기를 초과하는지를 체크한다.

- 사용자가 입력한 좌표에 돌이 이미 놓여있는지를 체크한다.

(4) 좌표에 O / X 놓기

1. 코드블록 스크린샷

```
// 4. 입력한 좌표에 현재 차례인 유저의 돌을 놓는다.
board[x][y] = currentUser;
```

2. 입력

   - currentUser => 현재 차례인 사용자의 돌

3. 결과

   - 좌표 x, y를 인덱스로하여 배열 요소에 돌을 저장

4. 설명

   - 입력한 좌표에 현재 차례인 사용자의 돌을 놓는다.

(5) 현재 보드판 출력

1. 코드블록 스크린샷

```
// 5. 현재의 보드판을 출력한다.
for (int i = 0; i < numCell; i++) {
    cout << "---|---|---" << endl;
    for (int j = 0; j < numCell; j++) {
        cout << board[i][j];
        // 각 행의 마지막 열의 돌이면 "|" 출력x
        if (j == numCell - 1) {
            break;
        }
        cout << "  |";
    }
    cout << endl;
}
cout << "---|---|---" << endl;
k++;
```

2. 입력

   - numCell => 보드판의 가로 및 세로 칸 개수

3. 결과

   - 보드판의 격자 무늬를 표현하는 문자열 출력

   - 이중 for-loop로 2차원 배열을 순회하며 각각의 요소를 출력

   - 각 요소를 출력한 후에는 "|"를 출력하여 칸 구분

   - 보드판 출력 후 k를 1증가

4. 설명

   - 현재 보드판에 놓인 돌들을 격자 무늬와 함께 문자열로 출력한다.

   - 보드판을 출력한 이후 k를 증가시켜 다음 사용자에게 차례를 넘긴다.

(6) 빙고 시 승자 출력 후 종료

1. 코드블럭 스크린샷

```
// 6. 현재의 보드판에 승자가 있다면 출력 후 종료한다.
char winner = ' '; // 승자가 있다면 해당 승자의 돌을 저장하는 변수
// 가로줄 검증
for (int i = 0; i < numCell; i++) {
    // 변수 user에 i번째 가로줄의 맨 왼쪽에 놓인 돌을 저장
    char user = board[i][0];
    // 해당 좌표에 돌이 없다면 그 가로줄은 더 이상 탐색할 필요 x, 다음 가로줄로 이동
    if (user == ' ') {
        continue;
    }
    // 가로줄을 오른쪽으로 한 칸씩 이동하며 user와 같은지 검증
    for (int j = 1; j < numCell; j++) {
        // 다르다면 해당 가로줄 탐색 종료
        if (board[i][j] != user) {
            break;
        }
        // 모든 가로줄의 돌이 user와 같다면 user가 승자
        if (j == numCell - 1) {
            winner = user;
        }
    }
    // 승자가 있다면 승자를 출력
    if (winner != ' ') {
        cout << "가로에 모두 돌이 놓였습니다!: ";
        if (winner == 'X') {
            cout << "1번 유저(X)의 승리입니다!" << endl;
        }
        else {
            cout << "2번 유저(O)의 승리입니다!" << endl;
        }
        cout << "종료합니다";
        break;
    }
}
// 이미 승부가 결정되었으므로, while-loop 종료
if (winner != ' ') {
    break;
}
```

```
// 세로줄 검증
for (int i = 0; i < numCell; i++) {
    // i번째 세로줄의 맨 위에 놓인 돌을 user에 저장
    char user = board[0][i];
    // 이하는 가로줄과 같은 로직으로 진행
    if (user == ' ') {
        continue;
    }
    for (int j = 1; j < numCell; j++) {
        if (board[j][i] != user) {
            break;
        }
        if (j == numCell - 1) {
            winner = user;
        }
    }
    if (winner != ' ') {
        cout << "세로에 모두 돌이 놓였습니다!: ";
        if (winner == 'X') {
            cout << "1번 유저(X)의 승리입니다!" << endl;
        }
        else {
            cout << "2번 유저(O)의 승리입니다!" << endl;
        }
        cout << "종료합니다";
        break;
    }
}
// 이미 승부가 결정되었으므로, while-loop 종료
if (winner != ' ') {
    break;
}
```

```
// 왼쪽 위 ~ 오른쪽 아래 대각선 검증
char user = board[0][0]; // 맨 왼쪽 위의 돌을 user에 저장
if (user != ' ') {
    // 대각선으로 좌표를 이동해가며 탐색
    for (int i = 1; i < numCell; i++) {
        // user와 다른 돌이 나오면 탐색 종료
        if (user != board[i][i]) {
            break;
        }
        // 대각선상 모든 돌이 user와 일치하면 승자
        if (i == numCell - 1) {
            winner = user;
        }
    }
}
// 승자가 존재하면 출력
if (winner != ' ') {
    cout << "왼쪽 위에서 오른쪽 아래 대각선으로 모두 돌이 놓였습니다!: ";
    if (winner == 'X') {
        cout << "1번 유저(X)의 승리입니다!" << endl;
    }
    else {
        cout << "2번 유저(O)의 승리입니다!" << endl;
    }
    cout << "종료합니다";
    break;
}
```

```
// 오른쪽 위 ~ 왼쪽 아래 대각선 검증
user = board[0][numCell - 1]; // 맨 오른쪽 위 돌을 user에 저장
if (user != ' ') {
    // 대각선으로 좌표를 이동해가며 탐색
    for (int i = 1; i <= numCell - 1; i++) {
        // user와 다른 돌이 있다면 탐색을 종료
        if (user != board[0 + i][numCell - 1 - i]) {
            break;
        }
        // 대각선상 모든 돌이 user와 일치하면 승자
        if (i == numCell - 1) {
            winner = user;
        }
    }
}
// 승자가 존재하면 출력
if (winner != ' ') {
    cout << "오른쪽 위에서 왼쪽 아래 대각선으로 모두 돌이 놓였습니다!: ";
    if (winner == 'X') {
        cout << "1번 유저(X)의 승리입니다!" << endl;
    }
    else {
        cout << "2번 유저(O)의 승리입니다!" << endl;
    }
    cout << "종료합니다";
    break;
}
```

2. 입력

    - numCell => 보드판의 가로/세로 칸 개수

    - winner => 승자의 돌을 저장해둘 변수

    - user => 탐색할 줄의 맨 앞의 돌

3. 결과

    - 가로, 세로 각 3줄과 양 대각선 상의 돌을 user와 비교

    - 줄을 탐색하던 도중 다른 돌이 나오면 탐색을 종료

    - 승자가 나오면 winner에 해당 돌을 저장하고 가로, 세로의 경우 다음 줄을 탐색할 필요가 없으므로, for문을 종료

    - winner에 저장해둔 승자의 돌을 출력하고 while문을 종료

4. 설명

    - 가로, 세로, 양 대각선 순서로 승자가 있는지를 탐색한다.

    - 탐색할 줄의 첫 번째 돌을 변수 user에 저장하고 이후의 돌들을 user와 비교하는 방식으로 승자가 있는지를 탐색한다.

(7) 모든 칸이 찼으면 종료

1. 코드블럭 스크린샷

```
// 7. 보드판이 꽉 찼는지를 검증하고 찼다면 종료
bool isFull = false; // 꽉 찼다면 true 아니면 false를 저장하는 변수
// 보드판 전체를 탐색
for (int i = 0; i < numCell; i++) {
    for (int j = 0; j < numCell; j++) {
        // 비어있는 칸이 있다면 탐색 종료
        if (board[i][j] == ' ') {
            break;
        }
        // 모든 칸이 차있다면 isFull에 true 저장
        if (i == numCell - 1 && j == numCell - 1) {
            isFull = true;
        }
    }
}
// 보드판이 꽉찼다면 출력 후 종료
if (isFull == true) {
    cout << "모든 칸이 다 찼습니다. 종료합니다";
    break;
}
```

2. 입력

- isFull => 보드판이 꽉찼는지에 대한 bool타입 변수
- numCell => 보드판의 가로/세로 칸 개수

3. 결과

- 보드판 전체를 탐색하며 비어있는 칸이 있으면 탐색 종료
- 비어있는 칸이 없다면 isFull에 true를 저장
- isFull이 true 즉, 보드판이 꽉 찼다면 출력 후 종료

4. 설명

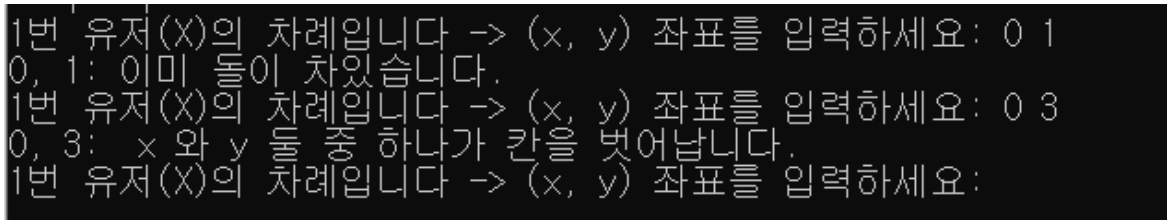
- bool 타입 변수를 도입해 보드판이 찼는지의 여부를 저장
- 이중 for문으로 보드판 전체를 탐색한다. 도중 빈 칸이 나오면 for문을 종료한다.

4. 테스트

1. 누구의 차례인지 출력

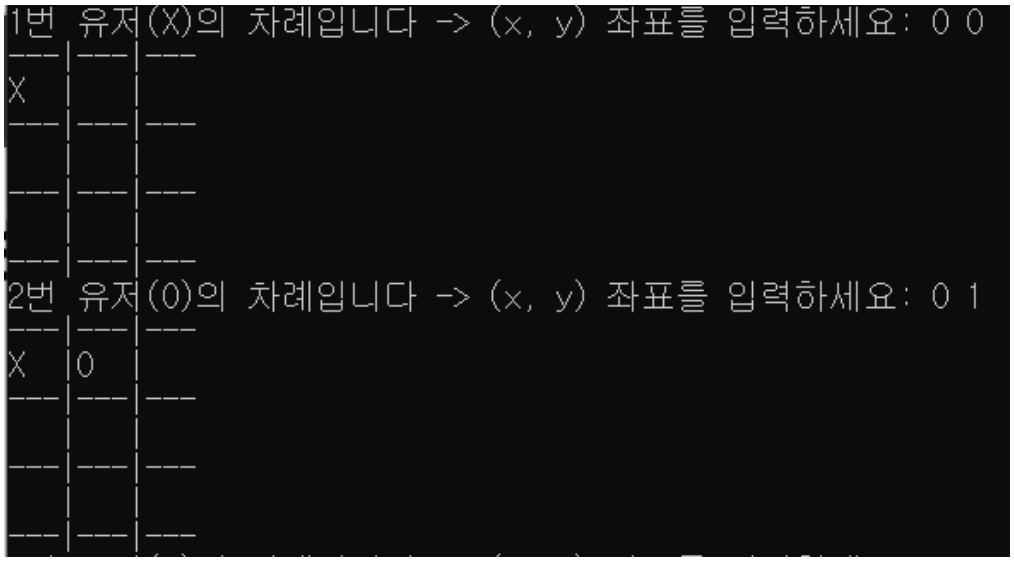
2. 좌표 입력 받기

3. 입력 받은 좌표 유효성 체크

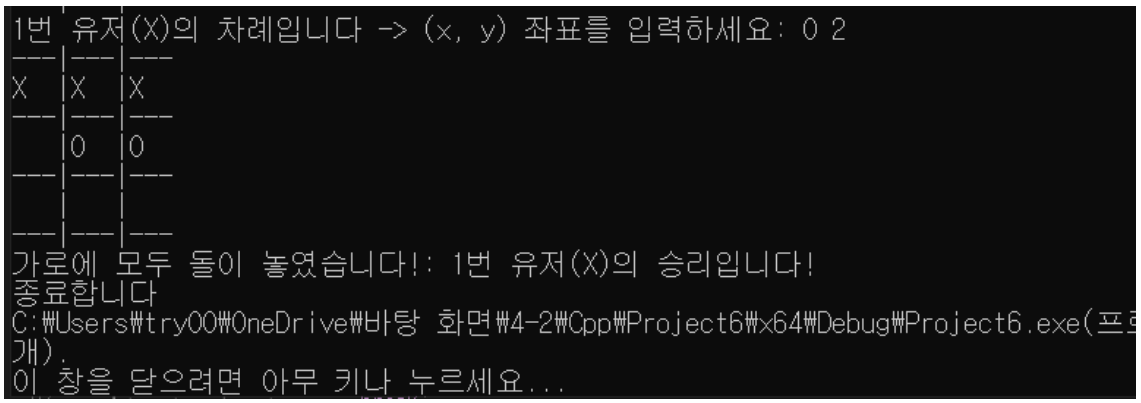


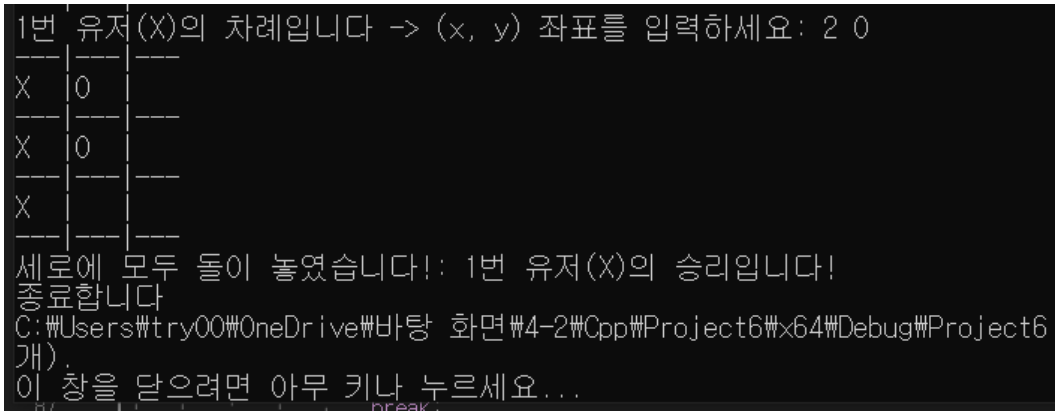
4. 좌표에 O / X 놓기

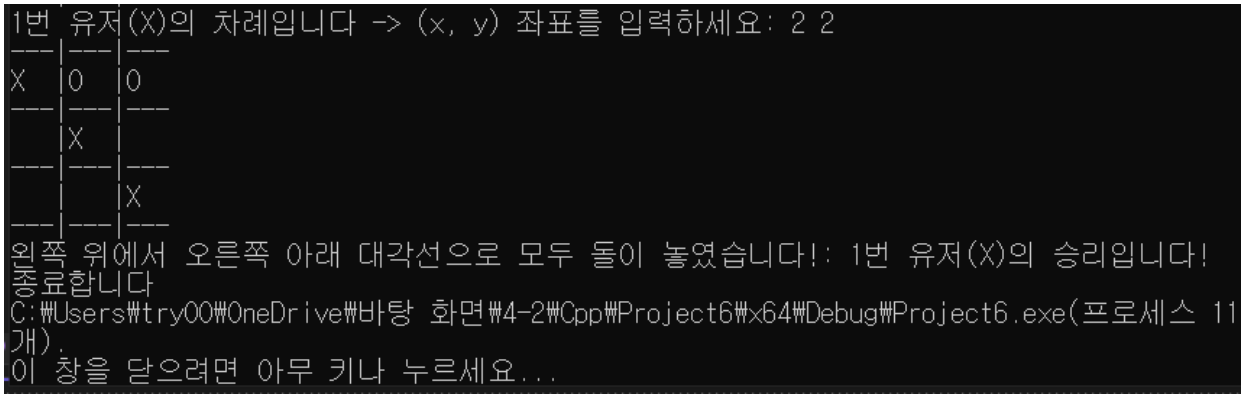
5. 현재 보드판 출력

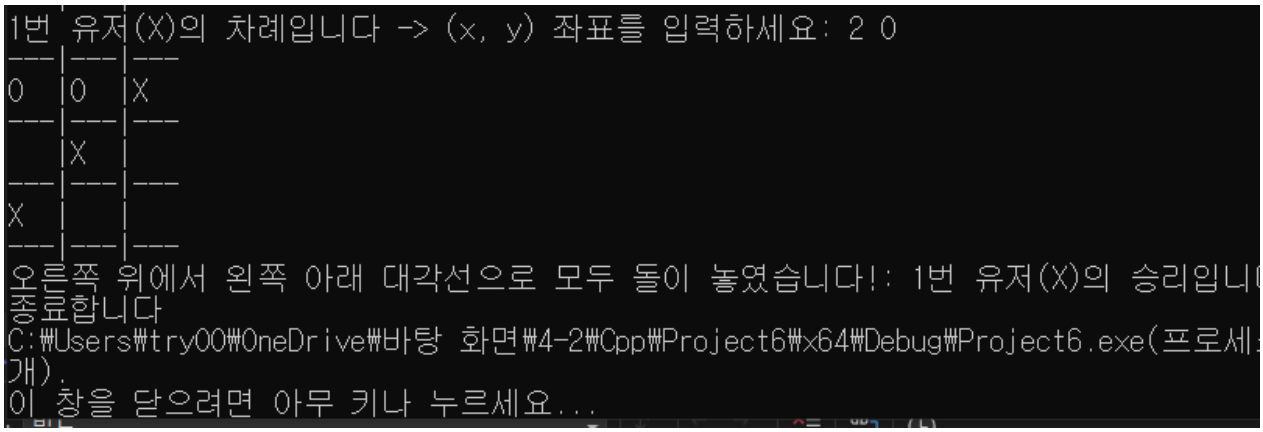


6. 빙고 시 승자 출력 후 종료

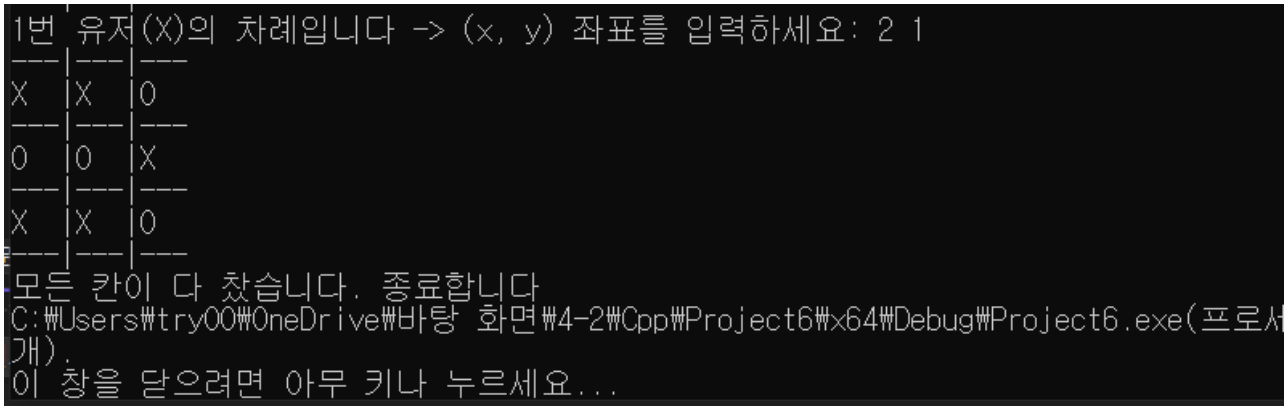



1번 유저(X)의 차례입니다 -> (x, y) 좌표를 입력하세요: 2 0
X |O |
X |O |
X | |
세로에 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!
종료합니다
C:₩Users₩try00₩OneDrive₩바탕 화면₩4-2₩Cpp₩Project6₩x64₩Debug₩Project6
개).
이 창을 닫으려면 아무 키나 누르세요...


1번 유저(X)의 차례입니다 -> (x, y) 좌표를 입력하세요: 2 2
X |O |O
|X |
| |X
왼쪽 위에서 오른쪽 아래 대각선으로 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니다!
종료합니다
C:₩Users₩try00₩OneDrive₩바탕 화면₩4-2₩Cpp₩Project6₩x64₩Debug₩Project6.exe(프로세스 11
개).
이 창을 닫으려면 아무 키나 누르세요...


1번 유저(X)의 차례입니다 -> (x, y) 좌표를 입력하세요: 2 0
O |O |X
|X |
X | |
오른쪽 위에서 왼쪽 아래 대각선으로 모두 돌이 놓였습니다!: 1번 유저(X)의 승리입니
종료합니다
C:₩Users₩try00₩OneDrive₩바탕 화면₩4-2₩Cpp₩Project6₩x64₩Debug₩Project6.exe(프로세
개).
이 창을 닫으려면 아무 키나 누르세요...

7. 모든 칸이 찼으면 종료


1번 유저(X)의 차례입니다 -> (x, y) 좌표를 입력하세요: 2 1
X |X |O
O |O |X
X |X |O
모든 칸이 다 찼습니다. 종료합니다
C:₩Users₩try00₩OneDrive₩바탕 화면₩4-2₩Cpp₩Project6₩x64₩Debug₩Project6.exe(프로세
개).
이 창을 닫으려면 아무 키나 누르세요...

5. 결과 및 결론

1. 프로젝트 결과 : 2명의 사용자가 진행하는 Tic Tac Toe 게임을 만들었다.

2. 느낀 점 : 프로젝트 문제의 정답이 너무 빨리 나와 알고리즘을 혼자 생각할 시간이 다소 부족했다.
   보고서를 작성하면서 코드를 다시 살펴볼 수 있어 디버깅하기가 수월했다.