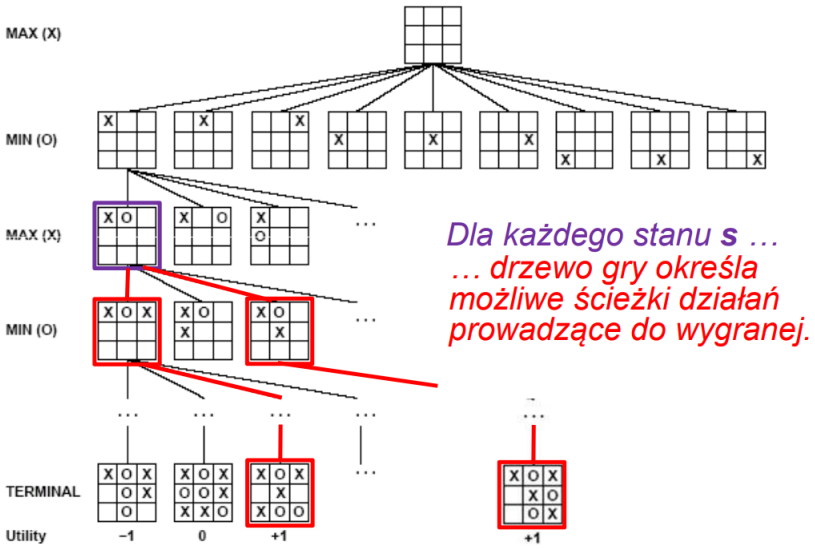
PAMSI	
Kierunek *Automatyka i Robotyka*	**Termin** *poniedziałek 15:15*
Imię, nazwisko, numer albumu *Artur Ziółkowski 259276*	**Data** *13 czerwca 2022*
Temat ćwiczenia *Projekt 3*	

1 Wstęp

Celem ćwiczenia było stworzenie działającej gry w kółko i krzyżyk, w której można wybrać rozmiar planszy i zadeklarować warunek zwycięstwa. W tym celu należało zaimplementować algorytm MinMax, aby umożliwić graczom grę przeciwko sztucznej inteligencji.

2 Algorytm MinMax

Idea działania algorytmu MinMax polega na tym, że dla każdego stanu, rekurencyjnie wywoływane są wszystkie możliwości rozgrywki. Następnie w zależności od osiągniętego stanu, nadawane są odpowiednim gałęziom punkty, a ruch który odpowiada największej ilości punktów jest wybierany przez AI. Poniżej pokazano schemat działania algorytmu.



Rysunek 1: Schemat działania algorytmu MinMax.

2.1 Implementacja algorytmu

W przygotowanym programie za działanie AI odpowiada klasa Game. Znajduje się w niej metoda get-best-move, która implementuje algorytm minimax. Punkty w algorytmie przyznawane sa za zwycięstwo w grze. Ponżej pokazano implementacje algorytmu.

```
template <uint16_t size, uint16_t win_condition>
Move Game<size, win_condition>::get_best_move(int player, int depth){

    end_game_conditiond rv = this->validate_win_condition();

    int blank_spaces = 0;
    for(int i = 0; i < size; ++i){
        for (int j = 0; j < size; ++j){
            if(this->board[i][j] == ' ') blank_spaces++;
        }
    }

    if(rv == AI_WIN){
        return Move(2*blank_spaces);
    }else if(rv == PLAYER_WIN){
        return Move(-2*blank_spaces);
    }else if (rv == TIE){
        return Move(0);
    }

    if (depth <= 0 ){
        return Move(0);
    }
    depth--;

    MyVector<Move> moves;
    // Do recursive
    for(int column = 0; column < size; ++column){
        for(int row = 0; row < size; ++row){
            if(this->board[row][column] == ' '){
                Move move;
                move.column = column;
                move.row = row;
                //board set value
                //if player is person
                if(player == 0){
                    this->board[row][column] = 'x';
                    // rv = this->validate_win_condition();
                    move.score = get_best_move(1, depth).score;
                    moves.push_back(move);
                    this->board[row][column] = ' ';
```

```
                }else{
                    this->board[row][column] = 'o';
                    // rv = this->validate_win_condition();
                    move.score = get_best_move(0, depth).score;
                    moves.push_back(move);
                    this->board[row][column] = ' ';
                }

            }
        }
    }
    // pick the best score
    int bestMove = 0;
    if(player == 1){
        int bestScore = INT16_MIN;
        for (int i = 0; i < moves.size(); ++i){
            if(moves[i].score > bestScore){
                bestMove = i;
                bestScore = moves[i].score;
            }
        }
    }else{
        int bestScore = INT16_MAX;
        for (int i = 0; i < moves.size(); ++i){
            if(moves[i].score < bestScore){
                bestMove = i;
                bestScore = moves[i].score;
            }
        }

    }
    return moves[bestMove];
}
```