확률 및 통계 Assignment 4

제목: Game Tree

과제 기간

2022년 5월 25일 (수)

~

2022년 4월 19일 (화)

학 과: 컴퓨터정보공학부

담당교수: 심동규 교수님

수업시간: 월, 수 5,6교시

학 번: 2018202065

성 명: 박 철 준

● 소개

1. 제목

Game Tree

2.목적

가. 게임 트리를 이해하고 이를 통해 Tic Tac Toe 게임을 구현하여야 한다.

나. 최적의 알고리즘을 찾고 이를 적용하여 게임 트리를 구성하여야 한다.

3. 프로그램 요구사항

가. “나”는 선공 플레이어를 맡고 “동생”는 후공 플레이어를 맡는다.

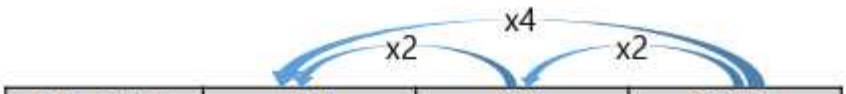
나. 선공 플레이어(나)를 위한 게임 트리를 구성한다.

다. 게임 트리에 존재하지 않는 state에 도달하면 “나”는 모든 칸에 동일한 확률로 수를 둔다.

라. 게임 트리를 구성하기 위해 5천번, 1만번의 시뮬레이션을 진행하였을 때와 랜덤하게 플레이 할 때의결과를 비교한다.

마. 결과 비교를 위한 게임 시뮬레이션은 100만회 이상 진행한다.

바. “동생”은 다음과 같이 규칙적인 확률로 수를 둔다.



현재 놓을 수 있는 수	귀에 수를 놓을 확률	변에 수를 놓을 확률	중앙에 수를 놓을 확률
귀, 변, 중앙	4/7	2/7	1/7
귀, 변	2/3	1/3	-
귀, 중앙	4/5	-	1/5
변, 중앙	-	2/3	1/3
귀	1	-	-
변	-	1	-
중앙	-	-	1

- 설계 및 구현

-접근 방법
프로그램 설계에 있어서 다음과 같은 시퀀스로 진행하였다. 진행하면서 발생한 이슈도 같이 설명하겠다.

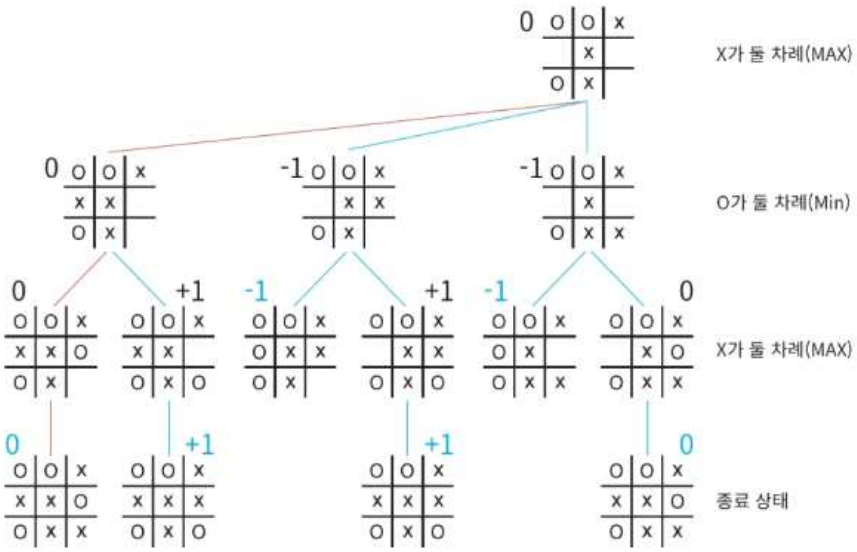
1. 1대1 방식의 tic-tac-toe 게임 구현
이는 가장 먼저 프로그램의 기초가 되는 tic-tac-toe 게임의 진행 방식을 이해하고 코드로 구현함으로 프로그램의 기초 프로토타입을 만든다.

2. 루프를 사용하여 2명의 자동 플레이어가 인터럽트가 발생하기 전까지 무한히 tic-tic-toe 게임이 진행되게 하여야 하였고 이때 수를 입력하는 방식은 rand 함수를 사용하여 두 플레이어의 입력하는 수는 random으로 설정하였다.

3. 한명의 자동 플레이어는 random의 수 대신 게임트리를 통해 학습한 데이터를 수로 이용하도록 설계하여야 하며 이때 최적의 게임트리를 구상하였고 minimax 알고리즘 서치 트리를 이용한 게임 트리로 결정하였다. 하지만 이 과정에서 약간의 문제 상황들이 발생하였다. 먼저 minimax 알고리즘의 서치 트리를 틱택톡 프로그램에 적용하기 위해서는 과연 어떤 정보들이 노드 구조체에 들어가야 하는지 감이 잡히지 않았다. 해서 탐색 알고리즘을 구현하다보면 자연스레 트리 노드의 value값에 대한 설정이 가능 할 것이라 생각되어 진다고 판단. 노드에 대한 정의는 후 순위로 다시 재조정하였다.

4. 구현해야 할 미니맥스 탐색 알고리즘의 설계하는 과정으로 돌입했다. 가장 먼저 일반적으로 미니맥스 알고리즘에 대한 공부를 시작했다.
Tic-Tac-Toe를 해결하기 위한 Minimax 알고리즘을 실행할 때
보드의 모든 미래 가능한 상태를 시각화하여 작동하고 이를 트리 형태로 구성한다.
만약 현재 보드 상태가 알고리즘(트리의 루트)에 주어질 때,
'n' 가지(여기서 n은 AI에 의해 선택될 수 있는 이동의 수/AI가 될 수 있는 빈 셀의 수를 나타냄)로 분할된다.
이러한 새 상태 중 하나가 최종 상태인 경우 이 상태에 대해 더 이상 분할이 수행되지 않고
다음과 같은 방식으로 점수가 할당됩니다.
점수 = +1 (AI가 이기는 경우)
점수 = -1 (AI가 지는 경우)
점수= 0 (무승부가 발생하는 경우)

- 틱택토 게임에서의 미니맥스 예시 -



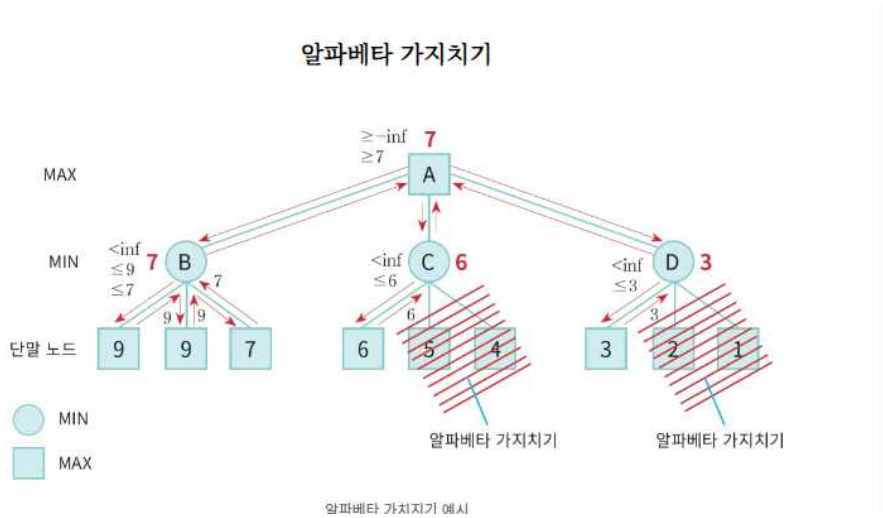
예를 들어 x가 이겨야하는 상황이라 가정할 때 깊이 4인 트리는 다음과 같다. 3번의 미래의 수를 보겠다는 뜻이고 보드의 다음 가능한 형태 모든 경우의 수 즉 depth를 m, 각 노드에서 만들 수 있는 자식 노드 수가 b라 할 때 O(b^m)만큼의 시간이 걸려 트리형태로 만들고 선택한다. 이때 선택의 방법은 상대방이 항상 최선의 수를 둔다는 가정하에, 할당한 점수를 반영하여 이겨야 하는 상황임으로 x의 차례 있어서는 점수가 높은 노드를 선택하고 o의 차례에 있어서는 점수가 낮은 것을 선택하는 방법으로 구현하면 된다.

다음은 수도 코드이다.

```
function minimax(node, depth, player)
    if depth == 0 or leaf node:
        return 해당 node의 heuristic 값
    if player == 'max':
        value = -(무한대)
        for each child of node:
            value = max(value, minimax(child, depth-1, 'min')
        return value
    else:
        value = +(무한대)
        for each child of node:
            value = min(value, minimax(child, depth-1, 'max')
        return value
```

방식은 위의 설명과 같다.

하지만 이렇게 미니맥스 탐색 알고리즘을 사용하여 최적한된 트리를 구하기 위해서 적어도 돌려야 하는 시뮬레이션은 258546번이 되기 때문에 5000번 혹은 만번의 시뮬레이션을 통해 올바른 트리를 구현하기도 힘들뿐더러 그러한 트리를 구현하면 값 또한 오차를 같게 되어 미니맥스 알고리즘 보다 더 효율이 좋은 알고리즘을 찾게 되었고 MINIMAX알고리즘에서 형성되는 탐색 트리 중에서 상당 부분은 결과에 영향을 주지 않으면서 가지들을 쳐낼 수 있다.
이것을 알파베타 가지치기라고 하는데 미니맥스 알고리즘으로 인한 최적화된 트리에 알파 베타 가지치기를 함께 적용한다면 최적화된 트리를 구현하는데 있어 하는 시뮬레이션을 5300번 ~ 5600번으로 훨씬 적게 할 수 있었다. 또한 이 이상으로 시뮬레이션을 하여도 트리는 이미 그 경우를 구현 하였음으로 시뮬레이션 수를 늘려줄 필요도 없다.



알파베타 가지치기 예시

방식은 그림과 같고 아래는 수도 해당 수도 코드이다.

```
function alphabeta(node, depth, alpha, beta, player)
    if depth == 0 or leaf node:
        return 해당 node의 heuristic 값
    if player == 'max':
        value = -(무한대)
        for each child of node:
            value = max(value, minimax(child, depth-1, 'min')
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = +(무한대)
        for each child of node:
            value = min(value, minimax(child, depth-1, 'max')
            beta = min(beta, value)
            if alpha >= beta:
                break
        return value
```

5. 노드 구조체를 정의하고 이러한 구조체에 노드 정보를 저장하는 것에 대한 이슈
상위 단계에서 트리를 구현함에 잇어서 따로 노드 구조체를 선언하고 이 구조체에 노드 정보를 저장하는 방식으로 구현해야하는 지에 대해 고민 하였고 이번 프로젝트에서 구현한 게임트리는 탐색 트리를 이용한 것임으로 노드 구조체에 대한 정의 없이 탐색 알고리즘을 통해 만나는 노드들을 따로 저장하지 않고 값을 비교하는 용도로만 사용할 수 있고 또한 노드를 구조체에 저장하여 루트 노드부터 탐색하는 알고리즘 자체가 이번 프로그램에서 구현한 알고리즘이고 저장하지 않고 해당하는 루트 노드 값을 뽑아내어 탐색하는 시간 복잡도가 이번 알파베타 가지치기를 이용한 미니맥스 알고리즘의 시간 복잡도 O(b^(m/2))와 같아서(같은 이유는 미니맥스 알고리즘은 재귀적 호출로 인한 노드 정보 추출이 관건인데 이는 노드를 저장하여 저장한 루트 노드에 접근하는 방식의 재귀적 호출이나 노드를 저장하지 않고 그때그때 추출되는 루트 노드에 대한 재귀적 호출과 같기 때문이다.) 따로 노드를 저장하는 구조체를 선언하여 저장하지 않았다.

6. 다음은 이러한 알파베타 가지치기 미니맥스 서치 트리 알고리즘을 사용하여 “나” 플레이어가 수를 둘수 있게 한다.

7. “동생” 플레이어가 수를 놓는 확률을 랜덤에서 과제 요구사항에 맞춰 현재 놓을 수 있는 수의 상황에 따라 놓는 확률을 달리하여 적용한다.

9. 결과를 도출한다.

- 함수 구현

다음은 해당 시퀀스를 진행하면서 구현한 함수들에 대한 설명이다.

1. 헤더와 전역변수

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define boardsize 3
#define true  1
#define false 0
#define me 1
#define brother    2
#define max(a, b) (((a) > (b)) ? (a) : (b))
#define min(a, b) (((a) < (b)) ? (a) : (b))

int checkcount_for_draw = 0; //이미 둔 자리의 수 9가 되면 비긴 것
int winner = 0; //me = 1, brother = 2
int maxDepth =8 ; //몇 수 앞까지 두어볼지를 결정 한수 앞만 고려
int bestPosition[3] = { 2, 2}; // Mimimax 함수에서 사용

int gui_arg[4] = { 0,2,6,8 };//귀
int byun_arg[4] = { 1,3,5,7 };//변
int center_arg[1] = { 4 };//중앙

int me_win = 0; //나의 승리 횟수
int brother_win = 0; // 동생의 승리 횟수
int draw = 0; // 무승부

int me_win_for_making_gametree = 0; //트리를 구성하기 위한 시뮬레이션에서 나의 승리
int brother_win_for_making_gametree = 0;//트리를 구성하기 위한 시뮬레이션에서 동생의 승리
int draw_for_making_gametree = 0;//트리를 구성하기 위한 시뮬레이션에서 무승부
int nomeaning_for_making_gametree = 0;//트리를 구성하기 위한 시뮬레이션에서 아직 승부가 결정되지 않은 경우

int minimax_tree_node_num = 0;//최적화 된 트리의 노드 개수 파악을 위해

//int random_counter = 0;
```

2. 좌표 설정 함수

```c
//1차원 숫자(1-9)로 입력된 값을 좌표 값으로 변환하기 위한 구조체
typedef struct tagPoint {
    int row;
    int col;
}Point;

// 1~9까지의 숫자를 3X3 2차원 배열에서 좌표로 변환
Point get_point_from_arr(int position)
{
    Point p;
    p.row = position / boardsize;
    p.col = position % boardsize;
    return p;
}

// 입력이 1차원이고, 보드는 2차원으로 되어 있어 입력된 위치에 체크되어 있는지 검사
int is_fill(int(*board)[boardsize], int position)
{
    Point p = get_point_from_arr(position);
    if (board[p.row][p.col])
    {
        return true;
    }
    return false;
}
```

3. initboard함수

```
61
62      // 보드를 0으로 초기화
63      void Initboard(int(*board)[boardsize])
64      {
65          int i, j;
66          for (i = 0; i < boardsize; i++)
67          {
68              for (j = 0; j < boardsize; j++)
69              {
70                  board[i][j] = 0;
71              }
72          }
73      }
```

보드를 초기화 한다. 보통 프로그램 초반에 사용하거나 시뮬레이션이 끝날 때마다 사용

4. set_number_about_borad 함수

```
105
106     // brother 또는 me로부터 입력된 값을 보드에 저장
107     void set_number_about_board(int(*board)[boardsize], int position, int player)
108     {
109         Point p = get_point_from_arr(position);
110         board[p.row][p.col] = player;
111         checkcount_for_draw++;
112     }
```

플레이어로부터 입력된 값을 보드에 저장

5. set_zero

```
114     // 주어진 위치를 0으로 복원
115     void set_zero(int(*board)[boardsize], int position)
116     {
117         Point p = get_point_from_arr(position);
118         board[p.row][p.col] = 0;
119         checkcount_for_draw--;
120     }
121
```

주어진 위치를 0으로 복원

6. is_win함수

```
// 승리체크
int is_win(int(*board)[boardsize], int player)
{
    if ((board[0][0] == player && board[0][0] == board[0][1] && board[0][1] == board[0][2]) || // 가로0
        (board[1][0] == player && board[1][0] == board[1][1] && board[1][1] == board[1][2]) || // 가로1
        (board[2][0] == player && board[2][0] == board[2][1] && board[2][1] == board[2][2]) || // 가로2
        (board[0][0] == player && board[0][0] == board[1][0] && board[1][0] == board[2][0]) || // 세로1
        (board[0][1] == player && board[0][1] == board[1][1] && board[1][1] == board[2][1]) || // 세로2
        (board[0][2] == player && board[0][2] == board[1][2] && board[1][2] == board[2][2]) || // 세로3
        (board[0][0] == player && board[0][0] == board[1][1] && board[1][1] == board[2][2]) || // 대각선1
        (board[2][0] == player && board[2][0] == board[1][1] && board[1][1] == board[0][2]))  // 대각선2
    {
        winner = player;
        return true;
    }

    return false;
}
```

게임 진행간에 승리를 체크한다.

7. evaluation 함수

```
// 게임이 끝이 났을 때의 점수
int evaluation()
{
    int score = 0;

    if (winner == me)
    {
        score = 1;
        me_win_for_making_gametree++;
    }
    else if (winner == brother)
    {
        score = -1;
        brother_win_for_making_gametree++;
    }
    winner = 0;
    return score;
}
```

트리를 구성하기 위한 시뮬레이션 과정에서 결과에 조사를 하는 함수

8. is_game_over 함수

```
// 누군가 승리를 했거나 비겼나 확인하고 결과를 출력
int is_game_over(int(*board)[boardsize], int player)
{
    char* Player[3] = { " ", "Compuer", "brother" };

    if (is_win(board, player))
    {
        //printf("%s won!\n", Player[player]);
        if (player == me)
        {
            me_win++;
        }
        else if (player == brother)
        {
            brother_win++;
        }

    }
    else if (checkcount_for_draw == boardsize * boardsize)
    {
        draw++;
        //printf("Draw!\n");
    }
    else
    {
        return false;
    }
    winner = 0;
    return true;
}
```

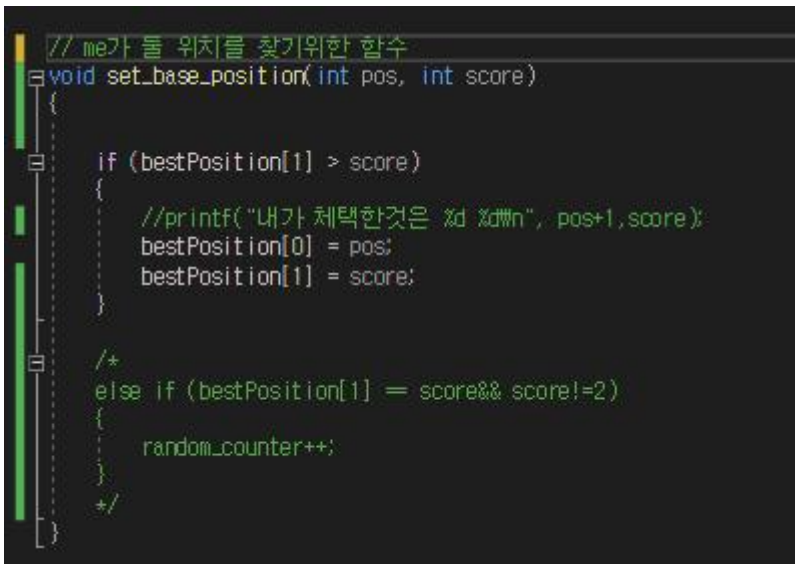
is_win함수로부터 트루 값이 들어 올 경우 게임이 끝난 것임으로 실행

9. get_empty_position 함수

```
// 보드의 빈곳을 위치값으로 변환하여 반환 배열의 끝에는 개수를 넣어준다.
void get_empty_position(int(*board)[boardsize], int* emptyPosition)
{
    int i, j, index = 0;
    for (i = 0; i < boardsize; i++)
    {
        for (j = 0; j < boardsize; j++)
        {
            if (board[i][j] == 0)
            {
                emptyPosition[index++] = i * boardsize + j;
            }
        }
    }
    emptyPosition[9] = index; //빈곳의 개수가 필요하므로 배열의 마지막에 표시해줌
}
```

보드의 빈곳을 위치값으로 변환하고 반환 배열의 끝에는 개수를 넣어준다.

10. set_base_position



```
// me가 둘 위치를 찾기위한 함수
void set_base_position(int pos, int score)
{

    if (bestPosition[1] > score)
    {
        //printf("내가 채택한것은 %d %d\n", pos+1,score);
        bestPosition[0] = pos;
        bestPosition[1] = score;
    }

    /*
    else if (bestPosition[1] == score&& score!=2)
    {
        random_counter++;
    }
    */
}
```

트리 구성간에 me가 둘 위치를 결정하는 함수

11. minmax_for_min_value 함수

```
// 베스트 위치를 찾기위한 서치함수
int minimax_for_min_value(int depth, int(*board)[boardsize], int player, int alpha, int beta)
{
    int win_check = is_win(board, 3 - player);

    if (depth == 0|| (win_check) || checkcount_for_draw == 9)
    {
        minimax_tree_node_num++;

        if(checkcount_for_draw == 9 && !(win_check))
        {
            draw_for_making_gametree++;

        }
        if (depth == 0 && !(win_check) && checkcount_for_draw != 9)
        {
            nomeaning_for_making_gametree++;
        }
        return evaluation();
    }
    int emptyposition[10];
    get_empty_position(board, emptyposition);
    int k = 0;
    if (player == me)
    {
        int i, score, minscore = 100;
        for (i = 0; i < emptyposition[9]; i++)
        {
            set_number_about_board(board, emptyposition[i], player);
            score = minimax_for_min_value(depth - 1, board, brother, alpha, beta);
            set_zero(board, emptyposition[i]);
            minscore = min(score, minscore);
            beta = min(minscore, beta);

            if (depth == maxDepth)
            {
                set_base_position(emptyposition[i], minscore);
            }
            if (beta <= alpha)
            {
                break;
            }
        }
        return minscore;
    }
```

```
    else
    {
        int i, score, maxscore = -100;
        for (i = 0; i < emptyposition[9]; i++)
        {
            set_number_about_board(board, emptyposition[i], player);
            score = minimax_for_min_value(depth - 1, board, me, alpha, beta);
            set_zero(board, emptyposition[i]);
            maxscore = max(score, maxscore);
            alpha = max(maxscore, alpha);
            if (beta <= alpha)
            {
                break;
            }
        }
        return maxscore;
    }
}
```

이번 과제에서 가장 중요한 알파베타 가지치기를 적용한 미니맥스 서치 트리 알고리즘 함수.

12. play_game 함수

```
// 게임이 진행되는 함수
int play_game(int(*board)[boardsize], int player)
{
    if (player == me)
    {

        minimax_for_min_value(maxDepth, board, player, -100, 100);
        // 한 수를 두고나면 베스트 값을 초기화해줘야 다음에 좋바른 값을 찾을 수 있슴
        bestPosition[1] =2;

        /* //트리를 구성할 때 존재하지 않는 스테이트에 경우에는 모든칸에 동일한 확률로 수를 둔다.
        //미니맥스 탐색 트리에서는 해당하는 수의 모든 모든 경우를 트리화 시킴으로 존재하지 않는 스테이트는 따로 존재하지 않는다.
        if ((emptyposition[9] - 1) == random_counter)
        {
            random_counter = 0;
            return getnumber_in_random(board);
        }
        else
        {
            random_counter = 0;
            return bestPosition[0];

        }
        */
        return bestPosition[0];
    }
    else
    {
        return get_number(board);
    }
}
```

게임을 진행하는 함수 이는 트리를 구성한 후 트리를 이용하여 최소 백만번의 시뮬을 돌릴 때 사용

13. playgame_in_random 함수

```
// 게임이 진행되는 함수 랜덤하게 시뮬레이션 할 때 사용
int playgame_in_random(int(*board)[boardsize], int player)
{
    if (player == me)
    {
        return getnumber_in_random(board);
    }
    else
    {
        return get_number(board);
    }
}
```

게임을 진행하는 함수이며 랜덤하게 시뮬레이션 할 때 사용

14. initgame 함수

```
// 게임이 시작될 때 초기화가 필요한 변수들을 모아서 초기화 해줌
int initGame(int(*board)[boardsize])
{
    int player;
    initboard(board);
    //print_board(board);
    checkcount_for_draw = 0;
    bestPosition[1] =2;
    player = 1; // 선을 선택하기 위하여
    return player;
}
```

게임을 시작할 때 초기화가 필요한 변수들을 모아서 초기화 한다.

15. get_number()함수

```
// 보드에 체크할 위치를 숫자로 입력받기위한 함수
int get_number(int(*board)[boardsize])
{
    //char number;
    int number=0;

    int emptyPosition[10];
    get_empty_position(board, emptyPosition);
    int gui = 0;
    int byun = 0;
    int center = 0;
    for (int i = 0; i < emptyPosition[9]; i++)
    {
        //printf("position = %d \n", emptyposition[i]);

        if ((emptyPosition[i] == 0) || (emptyPosition[i] == 2) || (emptyPosition[i] == 6) || (emptyPosition[i] == 8))
        {
            gui++;
        }
        if ((emptyPosition[i] == 1) || (emptyPosition[i] == 3) || (emptyPosition[i] == 5) || (emptyPosition[i] == 7))
        {
            byun++;
        }
        if ((emptyPosition[i] == 4))
        {
            center++;
        }
    }
```

```
    //printf("gui = %d \n", gui);//1
    //printf("byun = %d \n", byun);//2
    //printf("center = %d \n", center);//3
    int pro = 0;
    int next_position_type=0;
    if (gui != 0 && byun != 0 && center != 0)
    {
        pro = rand() % 7;

        if (pro >= 0 && pro <= 3)
        {
            next_position_type = 1;
        }
        else if (pro >= 4 && pro <= 5)
        {
            next_position_type = 2;
        }
        else if (pro == 6)
        {
            next_position_type = 3;
        }

    }
    else if (gui != 0 && byun != 0 && center == 0)
    {
        pro = rand() % 3;
        if (pro >= 0 && pro <= 1)
        {
            next_position_type = 1;
        }
        else if (pro == 2)
        {
            next_position_type = 2;
        }
    }
    else if (gui != 0 && byun == 0 && center != 0)
    {
        pro = rand() % 5;
        if (pro >= 0 && pro <= 3)
        {
            next_position_type = 1;
        }
        else if (pro == 4)
        {
            next_position_type = 3;
        }
    }
```

```
else if (gui == 0 && byun != 0 && center != 0)
{
    pro = rand() % 3;
    if (pro >= 0 && pro <=1)
    {
        next_position_type = 2;

    }
    else if (pro == 2)
    {
        next_position_type = 3;
    }
}
else if (gui != 0 && byun == 0 && center == 0)
{
    next_position_type = 1;
}
else if (gui == 0 && byun != 0 && center == 0)
{
    next_position_type = 2;
}
else if (gui == 0 && byun == 0 && center != 0)
{
    next_position_type = 3;
}
```

```
    while (true)
    {
        //printf("next_position_type = %d ₩n", next_position_type);//3
        if (next_position_type == 1)
        {

            number = rand() % 4;
            number = gui_arg[number];
            //printf(" 무한 루프 number = %d ₩n", number);//3
        }
        else if (next_position_type == 2)
        {
            number = rand() % 4;
            number = byun_arg[number];
        }
        else if (next_position_type == 3)
        {
            number = center_arg[0];
        }


        if (number < 0 || number > 8)
        {
            //printf("₩nPlease enter the correct number.₩n");
        }
        else if (is_fill(board, number))
        {
            //printf("Fill position, Please enter the other number₩n");
        }
        else
        {
            break;
        }
    }
    return number;
}
```

동생이 상황에 맞게 확률적으로 둘 수 있도록 설계한 함수 동생만 사용한다.

16. getnuber_in_random 함수

```
//랜덤하게 값을 뽑아내는 함수
int getnumber_in_random(int(*board)[boardsize])
{
    int number;
    while (true)
    {
        //printf("Which position do you want? Input number(1 ~ 9) : ₩n");
        number = rand() % 9;
        if (number < 0 || number > 8)
        {
            //printf("₩nPlease enter the correct number.₩n");
        }
        else if (is_fill(board, number))
        {
            //printf("Fill position, Please enter the other number₩n");
        }
        else
        {
            break;
        }
    }
    return number;
}
```

랜덤하게 시뮬레이션 할 때 “나” 가 사용하는 함수이다.

17. main 함수

```c
//메인 함수 게임 실행 등을 함
int main(void)
{
    int i;
    int player = 0;
    int position;
    int board[boardsize][boardsize];

    srand((unsigned)time(NULL));
    printf("₩n");
    printf("---------알파 베타 가지치기와 minimax 알고리즘을 이용하여 최대한 optimal한 게임트리를 구성하는 경우----------₩n");

    int k = 0;
    while (k < 1)
    {
        player = initGame(board);
        while (true)
        {
            position = play_game(board, player);
            set_number_about_board(board, position, player);
            //print_board(board);
            if (is_game_over(board, player))
            {
                break;
            }
            player = 3 - player;
        }
        k++;
    }
    int total_for_making_gametree = (me_win_for_making_gametree + brother_win_for_making_gametree + draw_for_making_gametree);
    printf("₩n");
    printf("게임트리를 구성하기 위해 %d번 시뮬레이션한 경우,₩n",total_for_making_gametree);
    printf("최대한 optimal한 트리를 구성하기 위한 시뮬레이션에서 ₩"나₩"의 승리는 %d번이다.₩n", me_win_for_making_gametree);
    printf("최대한 optimal한 트리를 구성하기 위한 시뮬레이션에서 ₩"동생₩"의 승리는 %d번이다.₩n", brother_win_for_making_gametree);
    printf("최대한 optimal한 트리를 구성하기 위한 시뮬레이션에서 ₩"나₩"와 ₩"동생₩"의 무승부는 %d번이다.₩n", draw_for_making_gametree);

    float per = (float)brother_win_for_making_gametree/total_for_making_gametree;
    printf("최대한 optimal한 노드를 구성하기 위한 시뮬레이션에서 ₩"동생₩"의 승률은 %f퍼센트이다.₩n", per*100);
```

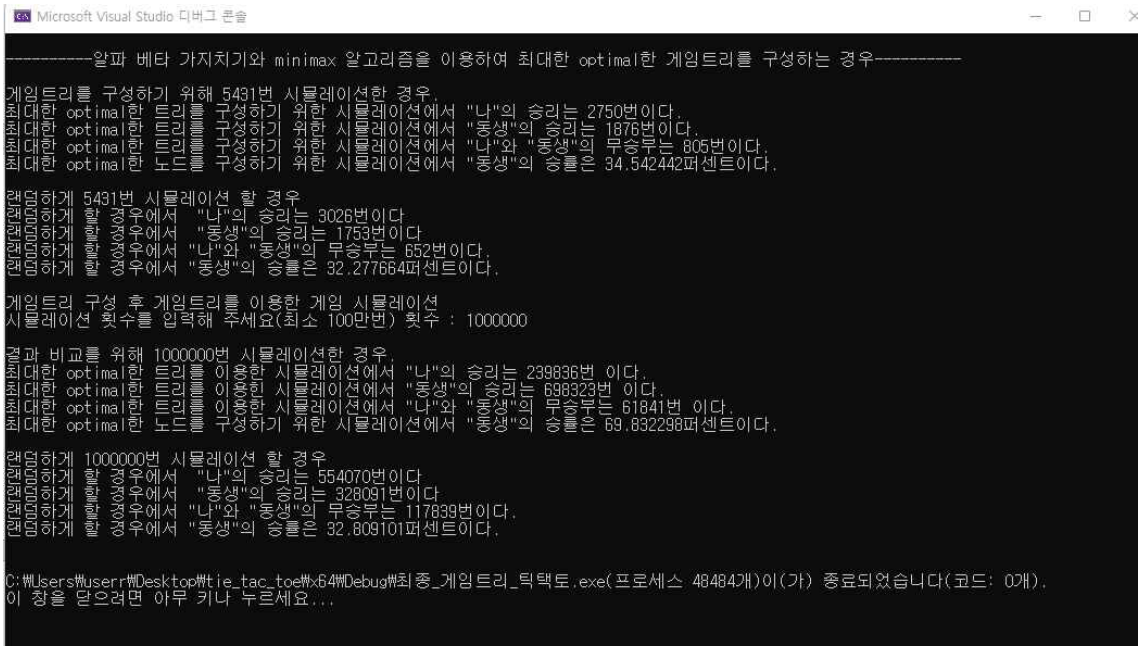
```c
    float per = (float)brother_win_for_making_gametree/total_for_making_gametree;
    printf("최대한 optimal한 노드를 구성하기 위한 시뮬레이션에서 ₩"동생₩"의 승률은 %f퍼센트이다.₩n", per*100);

    me_win = 0;
    brother_win = 0;
    draw = 0;
    printf("₩n");
    printf("랜덤하게 %d번 시뮬레이션 할 경우₩n",total_for_making_gametree);

    k = 0;
    while (k < total_for_making_gametree)
    {
        player = initGame(board);
        while (true)
        {
            position = playgame_in_random(board, player);
            set_number_about_board(board, position, player);
            //print_board(board);
            if (is_game_over(board, player))
            {

                break;


            }

            player = 3 - player;
        }

        k++;
    }
    printf("랜덤하게 할 경우에서  ₩"나₩"의 승리는 %d번이다 ₩n", me_win);
    printf("랜덤하게 할 경우에서  ₩"동생₩"의 승리는 %d번이다 ₩n", brother_win);
    printf("랜덤하게 할 경우에서 ₩"나₩"와 ₩"동생₩"의 무승부는 %d번이다.₩n", draw);
    int total_game = me_win + brother_win + draw;
    per = (float)brother_win / total_game;
    printf("랜덤하게 할 경우에서 ₩"동생₩"의 승률은 %f퍼센트이다.₩n", per * 100);
    me_win = 0;
    brother_win = 0;
    draw = 0;


    maxDepth = 1;
    printf("₩n");
    printf("게임트리 구성 후 게임트리를 이용한 게임 시뮬레이션₩n");
    me_win = 0;
    brother_win = 0;
    draw = 0;
```

```
    int num = 0;

    printf("시뮬레이션 횟수를 입력해 주세요(최소 100만번) 횟수 : ");
    scanf("%d", &num);
    printf("₩n");
    k = 0;
    while (k < num)
    {
        player = initGame(board);
        while (true)
        {
            position = play_game(board, player);
            set_number_about_board(board, position, player);
            //print_board(board);
            if (is_game_over(board, player))
            {
                break;
            }
            player = 3 - player;
        }
        k++;
    }
    printf("결과 비교를 위해 %d번 시뮬레이션한 경우.₩n",num);
    printf("최대한 optimal한 트리를 이용한 시뮬레이션에서 ₩"나₩"의 승리는 %d번 이다.₩n", me_win);
    printf("최대한 optimal한 트리를 이용한 시뮬레이션에서 ₩"동생₩"의 승리는 %d번 이다.₩n", brother_win);
    printf("최대한 optimal한 트리를 이용한 시뮬레이션에서 ₩"나₩"와 ₩"동생₩"의 무승부는 %d번 이다.₩n", draw);
    per = (float)brother_win / num;
    printf("최대한 optimal한 노드를 구성하기 위한 시뮬레이션에서 ₩"동생₩"의 승률은 %f 퍼이다.₩n", per * 100);
    total_game = brother_win + me_win + draw;
    me_win = 0;
    brother_win = 0;
    draw = 0;

    printf("₩n");
    printf("랜덤하게 %d번 시뮬레이션 할 경우₩n", num);
```

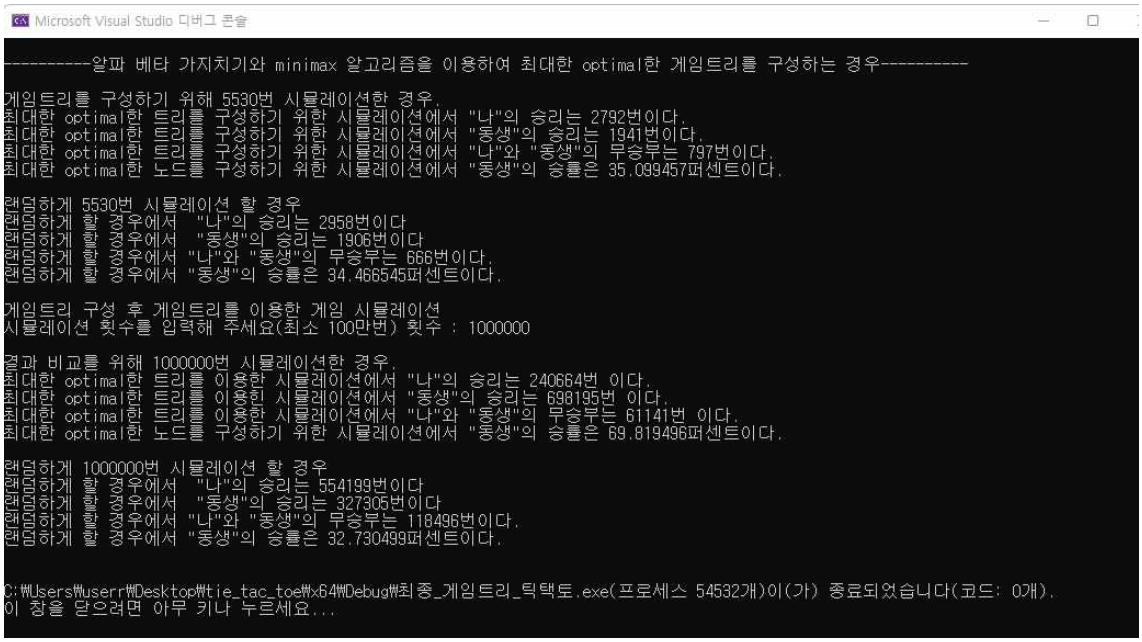
```
    k = 0;
    while (k < num)
    {
        player = initGame(board);
        while (true)
        {
            position = playgame_in_random(board, player);
            set_number_about_board(board, position, player);
            //print_board(board);
            if (is_game_over(board, player))
            {

                break;


            }

            player = 3 - player;
        }

        k++;
    }
    printf("랜덤하게 할 경우에서  ₩"나₩"의 승리는 %d번이다 ₩n", me_win);
    printf("랜덤하게 할 경우에서  ₩"동생₩"의 승리는 %d번이다 ₩n", brother_win);
    printf("랜덤하게 할 경우에서 ₩"나₩"와 ₩"동생₩"의 무승부는 %d번이다.₩n", draw);
    per = (float)brother_win / num;
    printf("랜덤하게 할 경우에서 ₩"동생₩"의 승률은 %f퍼센트이다.₩n", per * 100);
    me_win = 0;
    brother_win = 0;
    draw = 0;
    printf("₩n");

    return 0;

}
```

-SW 실행 결과

1. 첫 번째 실행



2. 두 번째 실행



두 결과를 비교하여 볼 때 최적의 트리를 구현하기 위한 시뮬레이션은 5300번에서 ~ 5600번 사이 까지 밖에 존재하지 않으며 이는 알파 베타 가지치기를 적용한 미니맥스 탐색 트리 알고리즘에서 깊이가 9인 즉 모든 경우를 탐색하여 얻은 트리이다. 이 이상으로 시뮬레이션은 되지 않는다. 만번의 시뮬레이션 불가능.

또한 트리를 구성하기 위한 시뮬레이션을 돌렸을 때 동생의 승률은 랜덤으로 시뮬레이션을 돌린것과 무방하기 때문에 즉 “게임 트리에 존재하지 않는 state에 도달하면 “나”는 모든 칸에 동일한 확률로 수를 둔다.”의 요구사항에 부합한다.
또한 최소 백만번의 시뮬레이션을 통해 얻은 동생의 승률은 랜덤으로 했을 때와 비교 했을 때 두배 정도 더 커진 것을 알 수 있다.

-고찰
이번 프로젝트를 진행하면서 겪은 의문점을 위주로 설명하고자 한다.
먼저 백만번의 진행을 통해 동생의 승률을 구하게 되면 확률적으로 계산하였을 시 최대로 옵티멀한 알고리즘으로 구현하였다고 가정할 시 동생이 이길 경우와 동생과 비길 경우만 발생할 것이라고 생각 되어졌지만 이번 과제를 진행한 결과 값에 있어서는 “나”가 이길 경우도 발생하였다. 이를 생각하여 볼 때 알파 베타 가지치기를 적용한 minimax 알고리즘에 있어서 약간에 설계 오류가 있던 것이 아닌가 생각 되며 또한 동생의 상황에 따라 놓는 수의 확률이 전 영역에 걸쳐 동등한 확률로 랜덤하게 나오는 것이 아닌기 때문이지 않을까 생각되어도 진다. 그래서 설계 오류를 찾아내는데 시간을 많이 들였지만 찾지 못하여 이를 해결하지 못한 것에 아쉬움이 생긴다.
또한 이러과정에 있어 다양한 게임트리 알고리즘에 대해 공부할 수 있었고 예를 들어 게임트리 알고리즘에 있어서는 minimax 알고리즘뿐만 아니라 몬테카를로 알고리즘 등에 대해서도 공부를 할 수 있었다. 이는 더 나아가 AI의 기본적인 기술인 기계학습에 대한 이해도를 이번 프로젝트 덕분에 높일 수 있었다.