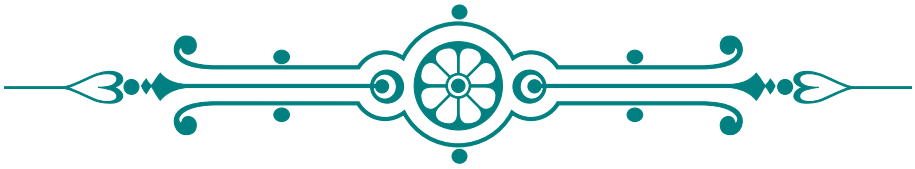


오목 인공지능 형성 설계과제 수행 계획서

교과목명 : 자료구조와 실습

담당교수 : 정 준호 교수님

프로젝트명 : Ω目

학번 : 2016110056

이름 : 박승원

날짜 : 2016년 11월 15일

차 례

제 1 장

설계과제 개요

제 1 절　과제 목표

최소한의 룰을 입력한 후, 스스로 학습하여 실력을 늘려가는 오목 게임 인공지능.

제 2 절　과제 내용

최근에 알파고라는 센세이셔널한 열풍이 휩쓸고 지나갔다. 컴퓨터가 스스로 대국을 두어가며, 실력을 늘려갈 수 있다는 아이디어는 매우 신선했다. 이에 그런 비슷한 프로그램을 한 번 만들어 보면 어떨까 하는 생각이 떠올랐다.

우선은 오목의 가장 기본 규칙인 다섯 개를 연달아 놓으면 이긴다는 가장 단순한 규칙을 주고, 그 이외에는 컴퓨터끼리 대국하여 실력을 늘려가는 방식을 차용하기로 했다.

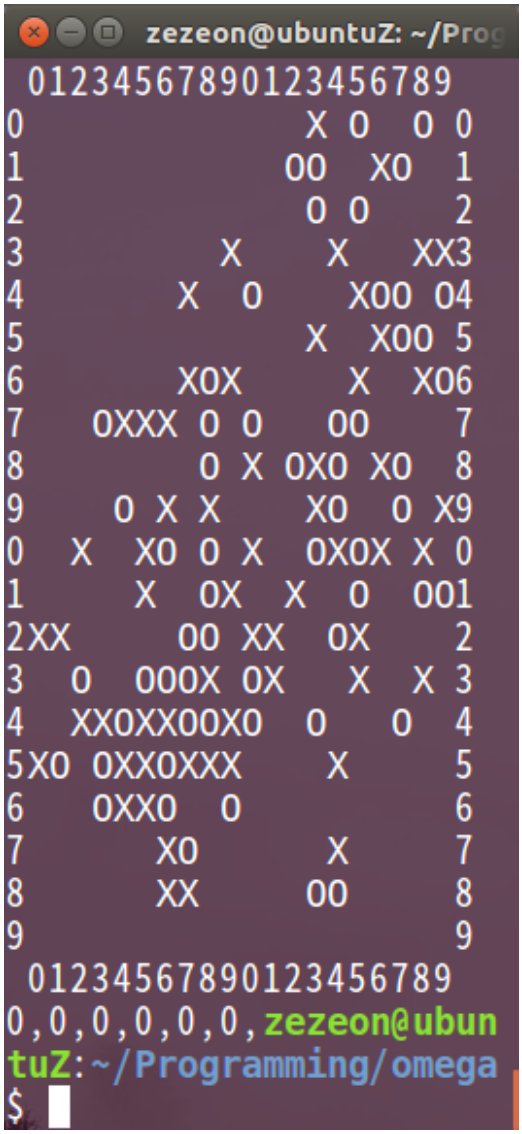
물론, 알파고의 인공 지능처럼 방대한 연산 능력이나, 고차원적인 알고리즘은 할 수 없지만, 그냥 단순히 기보를 기억하고, 그 승패의 결과를 누적해도 오목은 단순한 게임이기에 어느 정도의 인공지능은 달성할 수 있으리라 생각된다.

2.1　개발 설계

- char board[20][20]의 바둑판 위에 문자로 OX로 바둑돌을 표시한다.
- 바둑판을 분석하여 승패를 결정짓는 위치를 알아낼 수 있게 한다.
- 그러한 결정적인 위치가 없을 경우에는 현재 놓인 바둑돌의 2칸 이내의 랜덤한 위치에 바둑돌을 놓고 그 기보를 기억한다.
- 최종적으로 승패가 결정되었을 때에 그 판에 둔 모든 random한 바둑돌들의 데이터를 승패로 결정하여 누적한다.
- 컴퓨터끼리 많은 대국을 하여 데이터를 누적한다.
- 이렇게 누적한 데이터를 바탕으로 인간과 둘 때에, 각 위치의 승률을 따져서 높은 곳을 선택한다.

2.2 프로그램 구현

- 기보 데이터는 매우 많은 경우의 수가 있으므로, 데이터가 매우 크리라 생각된다. 그러므로, 기보 저장시 압축하는 알고리즘을 사용한다.
- 압축한 기보 데이터를 해싱과 이진 트리를 사용하여, 저장하고, 검색할 수 있게 한다.
- 텍스트로 보드를 표현한다.



2.3 마무리

- 어느 정도의 인공지능이 형성되었는지 평가한다.

제 3 절 과제 수행 방법

C언어로 프로그램을 작성하기로 했다. 혼자서 프로젝트를 진행하기로 하였으므로, 시간이 나는 틈틈이 프로그램을 짜기로 하였다.

제 2 장

설계과제 목표 및 주요 내용

제 1 절　과제 목표

학습형 오목 인공지능.

제 2 절　과제 내용

2.1　개발 설계

2.1.1　계획서 작성

현재까지는 바둑돌의 배열을 분석하여 결정적 위치를 찾는 것과, 임의의 위치에 바둑돌을 두는 것, 바둑판을 압축하여 표현하고 다시 원상 복구하는 것, 컴퓨터끼리 대국을 두는 것을 구현하였다.

기보를 승패의 결과에 따라 저장하고, 컴퓨터끼리 둔 대국의 데이터를 바탕으로 수를 결정하는 부분이 남았다. 이 부분은 해싱과 트리로 구현할 생각이므로, 시간이 좀 걸리리라 생각된다.

2.1.2　요구사항 검토

이 프로그램은 인공 지능 형성이 목적이므로, 텍스트로 평이하게 바둑판을 표현하고, 가로 세로 좌표를 콘솔로 입력하는 것으로 바둑돌을 두는 인터페이스를 선택하였다.

2.1.3　모듈 설계

- 현재의 바둑판을 압축하고 푸는 함수
- 오목 룰에 따라 승패를 결정하는 결정적인 포인트를 찾아내는 함수
- 랜덤한 위치에 바둑돌을 두는 함수
- 기보를 해싱과 이진트리를 이용하여 저장하고 검색하는 함수
- 컴퓨터끼리 대국을 두거나 인간을 상대로 두는 인공 지능 함수

2.2 프로그램 구현

Listing 2.1: 기보를 압축하고 푸는 함수

```
#include<stdio.h>
#include<stdlib.h>
#define HEADER_SIZE 6
extern char board[20][20];

char* compress() {//현재의 보드를 압축
   int n = 0, i = 0;
   char b[400];
   for(int y=0; y<20; y++) for(int x=0; x<20; x++) {
      n++;
      if(n == 128) {//128개의 space가 연달아 올경우 0을  집어넣는다 .
         b[i++] = 0;
         n = 0;
      }
      if(board[y][x] == 'O') {//그 전의 O또는 X에서의 거리만큼  값으로
         b[i++] = n;
         n = 0;
      } else if(board[y][x] == 'X') {
         b[i++] = -n;
         n = 0;
      }
   }
   char *r = (char*)malloc(i+ HEADER_SIZE);
   for(int j=0; j<i; j++) r[j+HEADER_SIZE] = b[j];
   r[0] = i /100;
   r[1] = i %100;
   return r;// 첫두바이트는  사이즈 , 다음  두바이트는  공란 (nth v)다음
      2vcount 다음부터  데이터
}

void decompress(char* data) {//압축된 데이터를 바둑판에 다시  편다 .
   int sz = data[0] *100 + data[1];
   int n = 0, i = HEADER_SIZE;

   for(int y=0; y<20; y++) for(int x=0; x<20; x++) {
      n++;
      if(n == data[i]) {
```

```
            board[y][x] = 'O';
            n = 0;
            i++;
        } else if(n == -data[i]) {
            board[y][x] = 'X';
            n = 0;
            i++;
        } else if(n == 128) {
            i++;
            n = 0;
        }
        if(i == sz+HEADER_SIZE) break;
    }
}
```

Listing 2.2: find_straight() 바둑판에서 원하는 바둑돌의 배열을 찾아내는 함수

```
#include<string.h>
#include"queue.h"
extern char board[20][20];
static Queue queue;

int find_straight(const char* str) {//" sOOO " -> s : return space
    char st[10] = {};
    strcpy(st, str);
    int rx = 0, ry = 0, r;
    int length, i;
    for(i=0; st[i] != '\0'; i++) {
        if(st[i] == 's') {
            r = i;//record r position
            st[i] = ' ';//change to space
        }
    }
    length = i;

    qinit(&queue);//W
    for(int y=0; y<20; y++) {
        qinit(&queue);
        for(int x=0; x<20; x++) {
            while(!qcompare(&queue, st)) qpop(&queue);//pop until
```

```
            queue == st
         if(qcompare(&queue, st) == -1) qinsert(&queue, board[y][x
            ]);
         if(qcompare(&queue, st) == 1) {
            rx = x - length + r;
            ry = y;
            return rx * 100 + ry;
         }
      }
   }
   qinit(&queue);//N
   for(int x=0; x<20; x++) {
      qinit(&queue);
      for(int y=0; y<20; y++) {
         while(!qcompare(&queue, st)) qpop(&queue);//pop until
            queue == st
         if(qcompare(&queue, st) == -1) qinsert(&queue, board[y][x
            ]);
         if(qcompare(&queue, st) == 1) {
            rx = x;
            ry = y - length + r;
            return rx * 100 + ry;
         }
      }
   }
   qinit(&queue);//NW
   for(int x=0; x<20; x++) {
      qinit(&queue);
      for(int y=0; y<20 && x+y < 20; y++) {
         while(!qcompare(&queue, st)) qpop(&queue);//pop until
            queue == st
         if(qcompare(&queue, st) == -1) qinsert(&queue, board[y][x+
            y]);
         if(qcompare(&queue, st) == 1) {
            rx = x +y - length + r;
            ry = y - length + r;
            return rx * 100 + ry;
         }
      }
```

```
    }
    qinit(&queue);
    for(int y=0; y<20; y++) {
        qinit(&queue);
        for(int x=0; x<20 && x+y < 20; x++) {
            while(!qcompare(&queue, st)) qpop(&queue);//pop until
                queue == st
            if(qcompare(&queue, st) == -1) qinsert(&queue, board[x+y][
                x]);
            if(qcompare(&queue, st) == 1) {
                rx = x - length + r;
                ry = x + y - length + r;
                return rx * 100 + ry;
            }
        }
    }
    qinit(&queue);//NE
    for(int x=0; x<20; x++) {
        qinit(&queue);
        for(int y=0; y<20 && x-y >= 0; y++) {
            //printf("(%d,%d)", x-y, y);
            while(!qcompare(&queue, st)) qpop(&queue);//pop until
                queue == st
            if(qcompare(&queue, st) == -1) qinsert(&queue, board[y][x-
                y]);
            if(qcompare(&queue, st) == 1) {
                rx = x - y + length - r;
                ry = y - length + r;
                return rx * 100 + ry;
            }
        }
    }
    qinit(&queue);
    for(int y=0; y<20; y++) {
        qinit(&queue);
        for(int x=19; y<20 && x >=0; x--) {
            //printf("(%d,%d)", x,19-x+y);
            while(!qcompare(&queue, st)) qpop(&queue);//pop until
                queue == st
```

```
            if(qcompare(&queue, st) == -1) qinsert(&queue, board[19-x+
               y][x]);
            if(qcompare(&queue, st) == 1) {
               rx = x + length - r;
               ry = 19 - x + y - length + r;
               return rx * 100 + ry;
            }
         }
      }
      return -1;
}
```

Listing 2.3: 결정적 위치가 있을 경우 그 위치를 선택하고, 없을 경우에 랜덤한 위치를 선택하는 원초적 인공지능 함수, 이렇게 누적한 데이터를 바탕으로 나중에 좀 더 세련된 인공지능을 형성한다.

```
#include<ctype.h>
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
typedef char element;
typedef struct Tree {
   element* data;
   unsigned int* result;
   struct Tree *left, *right;
} Tree;
int find_straight(const char* s);
void free_tree(Tree*);
Tree* tinsert(Tree* p, element* data, int win);
char* compress();

char board[20][20];
struct Tree* tree = NULL;

static const char *win_string[] = {
   "sOOOO", "OsOOO", "OOsOO", "OOOsO", "OOOOs",//승리O
   "sXXXX", "XsXXX", "XXsXX", "XXXsX", "XXXXs",
   " sOOO ", " OsOO ", " OOsO ", " OOOs ", "O OsO O",//결정적
   " sXXX ", " XsXX ", " XXsX ", " XXXs ", "X XsX X"
};
char* Ogibo[200];//static or ={}선언시 실행시 크래시 ???
```

```
char* Xgibo[200];

void win(char ox) {
   for(int i=0; Ogibo[i]; i++) {
      //save
      tree = tinsert(tree, Ogibo[i], ox == 'O');
      free(Ogibo[i]);
      Ogibo[i] = NULL;
   }
   for(int i=0; Xgibo[i]; i++) {
      tree = tinsert(tree, Xgibo[i], ox == 'X');
      free(Xgibo[i]);
      Xgibo[i] = NULL;
   }
}

int check() {//2칸 이내에 v마크를 하고 v마크의 개수를 리턴
   int r = 0;
   for(int y=0; y<20; y++) for(int x=0; x<20; x++) {
      if(board[y][x] == 'O' || board[y][x] == 'X') {
         for(int y2=y-2; y2 <= y+2; y2++) for(int x2=x-2; x2 <= x
            +2; x2++) {
            if(y2>=0 && y2<20 && x2>=0 && x2<20 && board[y2][x2] ==
                ' ') {
               board[y2][x2] = 'v';
               r++;
            }
         }
      }
   }
   return r;
}

int record(char* gibo[]) {
   int i=0;
   while(gibo[i]) i++;
   char* cp = gibo[i] = compress();//현 상태의 기보를 저장.
   int count_v = check();
   int v = rand() % count_v;//두 칸 내에서 랜덤으로 고른후
```

```
   gibo[i][2] = v / 100;//랜덤 값을 저장한다.
   gibo[i][3] = v % 100;
   gibo[i][4] = count_v / 100;
   gibo[i][5] = count_v % 100;
   return v;
}

int put(int n, char ox) {//v마크를 지우고 n번째 v마크에  바둑돌을   둔다 .
   int i = 0, r;
   for(int y=0; y<20; y++) for(int x=0; x<20; x++) {
      if(board[y][x] == 'v') {
         if(i == n) {
            board[y][x] = ox;
            r = 100 * x + y;
         } else board[y][x] = ' ';
         i++;
      }
   }
   return r;
}

int Oai() {
   int xy, i;
   for(i=0; i<20; i++) {
      xy = find_straight(win_string[i]);
      if(xy != -1) {
         board[xy%100][xy/100] = 'O';
         break;
      }
   }
   if(i<5) {
      win('O');
      return xy;// 이 위로는  필연적인   룰에  따라
   }
   i = 0;//reuse
   if(xy == -1) put(record(Ogibo), 'O');//랜덤으로 고르는 부분
   return -1;
}
```

```
int Xai() {
   int xy, i;
   for(i=9; i>=0; i--) {
      xy = find_straight(win_string[i]);
      if(xy != -1) {
         board[xy%100][xy/100] = 'X';
         break;
      }
   }
   if(i>4) {
      win('X');
      return xy;
   }
   for(i=19; i>=10; i--) {
      xy = find_straight(win_string[i]);
      if(xy != -1) {
         board[xy%100][xy/100] = 'X';
         break;
      }
   }//이 위로는 필연적인 룰에 따라 고르는 부분
   i = 0;
   if(xy == -1) put(record(Xgibo), 'X');//랜덤으로 고르는 부분
   return -1;
}
```

2.3 마무리

- 인공지능의 수준을 평가한다.
- 개선할 만한 사항을 살펴본다.
- 더 나은 알고리즘에 대해 생각해 본다.

제 3 절 과제 수행 방법

- C언어로 전적으로 작성하기로 하였다.
- 모듈화를 최대한 하여 버그를 줄이도록 노력한다.
- 큐, 트리, 해싱 등의 자료구조를 최대한 활용한다.

제 3 장

과제 수행 일정

세부 개발내용	주별 세부 추진 일정
자료수집 및 설계 계획 수립	~ 11/16
요구사항 검토	11.17
구조도 및 모듈 설계	11.18
프로그램 설계	11.19
프로그램 코딩	11.20~ 11.25
성능확인 및 오류 수정	11.26~ 11.30
최종 점검 및 개선 사항 보완	12.1