Report

자료 구조
Term Project
최종 보고서

Implementation of Board Game ‘DA VINCI-CODE’

2014140419 식품공학과 김윤성
2016250053 바이오의공학부 김은채
2017320132 컴퓨터학과 장연석
2017320243 컴퓨터학과 정호용

CONTENTS

1. 서론

1.1. DA VINCI-CODE Board game 선정이유

다빈치 코드(Da Vinci Code) 보드 게임은 블록의 오름차순 배열을 이용한 추리 게임이다. 논리적인 추리, 심리전, 운의 3 가지 요소가 작용하며, 게임에서 승리하기 위해서 플레이어는 논리적인 판단, 기억력, 그리고 상대방을 속이는 약간의 센스가 필요하다. 자료구조 수업을 통해 리스트, 스택, 큐, 트리, 정렬, 그래프를 배웠고, 수업시간에 배운 자료구조 내용을 효율적으로 적용하여 다빈치 코드 보드게임을 구현할 수 있을 것이라 판단하였다.

예를 들어 플레이어의 블록을 오름차순으로 배열하는 것에 이중연결리스트를 활용하거나, 블록 더미에서 흰색과 검은색 블록 중 하나를 선택하여 랜덤하게 뽑는 것은 덱(deque)를 활용할 수 있을 것이라는 것 등이 팀플 첫 회의에서 다빈치 코드 보드게임을 이번 term project 주제로 선정한 것이 주요 이유였다.

우리 조의 최종 목표는 Bluffing 을 할 가능성이 있는 User(사람)와 플레이가 가능한(유의미한 승률을 보이는) AI 를 만들어 Da Vinci Code game 의 구현이다. 즉, 다빈치코드 게임의 기본 룰을 시행하는 알고리즘과 User 와 플레이를 하는 컴퓨터를 구현하는 알고리즘이 필요하다.

1.2. DA VINCI-CODE 개발 내용 Introduction

본론에서 system architecture 를 구현하기 위하여 DA VINCI-CODE 게임의 전반적인 플레이에 대한 이해가 필요하다. 다빈치 코드에서는 다음의 게임 요소들로 이루어 진다.

0~11 까지의 숫자, (-)가 그려져 있는 흰색 (13 장) (이때의 (-)카드는 조커 로 작용)
0~11 까지의 숫자, (-)가 그려져 있는 검정색 (13 장)
플레이어 (2~4 명)

블록들을 무작위로 섞은 후 플레이어들은 총 26 개의 블록 중 4 개씩 랜덤으로 가져와서 다른 플레이어들은 보지 못하고 자신만 볼 수 있도록 세워 둔다. 세워둘 때는 오름차순으로 숫자를 배열해야 하며, 같은 숫자일 경우에는 검정색이 왼쪽, 흰색 블록이 오른쪽에 오도록 배치한다. 조커 블록은 원하는 자리 어느 곳이든 자리에 놓을 수 있다. 남은 블록들은 '더미'에 모아둔다.

게임 준비가 완료되면 게임방법은 다음과 같다.

① 자신의 차례가 되면 블록더미 가운데 하나를 가져와 이미 세워놓은 타일 사이에 오름차순(조커는 원하는 위치, 같은 숫자는 검정이 앞쪽)에 맞게 세워둔다.
② 그리고 나서 다른 사람의 타일 가운데 하나를 가리키며 어떤 숫자가 적혀 있는지 맞히는 방식으로 다른 사람의 코드를 추리한다.
③ 만약, 추리가 틀렸을 경우에는 정답을 틀린 플레이어가 더미에서 가져왔던 블록을 눕혀 모두에게 공개한다. 그리고 다음 사람에게 차례를 넘긴다.
④ 만약, 추리가 맞았다면 상대방은 지목되어 들킨 블록을 눕혀 모두에게 공개한다.
⑤ 계속해서 상대방의 블록을 추리할 지, 차례를 다음 플레이어에게 넘길지 선택할 수 있다.
⑥ 계속해서 추리를 할 경우 블록을 한 번 더 추리한다. 다만, 한 번이라도 추리가 틀리면 이번 차례에 가져왔던 블록을 공개하고 차례를 넘겨야 한다.
⑦ 추리를 계속하지 않고 차례를 넘길 경우에는 이번 차례에 가져온 블록을 공개하지 않는다. 따라서 내가 가진 코드에 대한 정보를 덜 노출할 수 있다.
⑧플레이어끼리 돌아가면서 위의 방식을 반복하여 상대편의 숫자 배열을 다 맞춘 플레이어가 승리하게 된다.

게임의 전반적인 룰을 이해하였기에, 이제 어떻게 코드로 구현할 것인지 계획할 수 있었다. 그 내용은 본론 부분에서 다루기로 한다.
AI 가 decision making 을 하기 위해서는 게임 승리전략에 대한 파악도 필요하였고,
다빈치보드게임에서의 승리전략은 다음과 같다.

① 오름차순으로 수가 배열되어 있음을 알고 있으므로 공개된 블록의 앞뒤를 통해 수의 범위를 줄일 수 있게 된다.
② 다른 사람의 블록을 지목할 때, 대부분 자신에게 없는 숫자의 블록을 물어보게 되므로, 상대방이 물어본 숫자의 블록이 상대방에게 없다고 가정해 추리한다.
③ bluffing 을 하는 유저일 경우, ②을 통해 상대방이 추리할 것을 고려하여 자신에게 있는 숫자 블록도 말하여 혼란을 주기 때문에 ②번을 100% 신뢰해서는 안 된다.
④ 조커를 뽑았을 때, 어떤 숫자 사이에 배치할지도 중요한 요소이다. 상황에 따라 다르나, 일반적으로는 같은 숫자 두 블록(예를 들면 검정 4 와 흰색 4)사이에 배치하거나, 연속된 숫자 배열(예로, 검정 4 와 검정 5)사이에 배치하면 상대방이 블록을 추리하기 어렵게 할 수 있다.

각각의 승리전략에 대하여 AI decision making 을 어떻게 구성할 것인지 본론에서 다루기로 한다.

2. 기존의 DA VINCI-CODE game 과의 차별성

2.1 기존 게임 조사

Da Vinci-Code board game 을 구현하기 앞서, 시중에 출시된 다빈치 보드게임 앱이 있는지 구글 플레이스토어에서 검색하여 보았다. 검색한 결과, 'Enigma'라는 다빈치 보드게임 앱을 확인할 수 있었다. 'Enigma'를 분석한 결과, 다음과 같았다.

싱글플레이가 가능하며(이 경우 컴퓨터와 대결), 유저 2~4 명으로 플레이를 구성할 수 있었다. 유저는 자신의 핸드폰 앱을 이용하여 네트워크로 플레이를 주고받을 수 있었다.

싱글플레이에서의 컴퓨터와의 대결은 유저의 레벨에 따라 컴퓨터의 승률이 더 높아지도록 설정한 듯 보였으며, 이 경우 컴퓨터의 추측 정확도가 높았으나 정확히 어떠한 알고리즘을 가지고 추측하는 것인지는 확인할 수 없었다. 그러나 유저의 레벨이 낮았을 때, 유저에게 지목한 숫자를 또 지목하는 등 일반적인 유저(사람)이라면 하지 않을 추측을 하는 것을 보아 알고리즘의 수준이 높다고는 판단되지 않았다.

또한, bluffing 을 시도하였을 때, 유저(사람)이 부른 숫자의 블럭을 기억한다면 신뢰도를 고려하였을 때, AI 의 승률이 감소한다거나, 또는 bluffing 을 알아차려 AI 의 승률을 일정이상 유지하려는 듯한 모습이 전혀 보이지 않은 것을 보아 유저의 블럭을 기억하는 알고리즘은 사용하지 않은 듯하였다.

다만, 같은 숫자의 두 블럭(검정 6 과 흰색 6)사이에 흰색의 조커를 넣은 것을 게임을 통해 확인하였기에 조커를 유저가 헷갈리는 위치에 넣는 전략은 사용한 것으로 보인다.

2.2. 'Enigma'와의 차별성

'Enigma'는 여러 명의 유저(최대 4 명)이 게임플레이를 진행할 수 있고, 그래픽적인 면과 유저의 레벨, 출석 체크 도장 같은 카테고리 면에서는 우리가 진행하고 있는 term project 보다 완성도가 높지만, Davinci Code 보드게임의 핵심이라 할 수 있는 '논리적인 추리', '약간의 속임수(bluffing)'을 고려하지 않았다.

다빈치 코드 보드게임의 전체적인 게임방식과 승리전략을 면밀히 관찰, 분석한 결과 유저와 게임을 하게 되는 AI 또한 논리적인 추리와, 유저의 속임수를 가려서 판단하는 능력을 필요로 한다고 결론지었다.

AI decision making 구현 과정에서, 유저를 신뢰할 수 있는지 신뢰도(reliability) 개념을 도입하였고, 유저가 블러핑을 한 경우에는 유저에 대한 신뢰도를 감소시키는 방향으로 구현하였다. 또한 큐(queue)를 응용하여 유저의 선택을 기억하는 AI 의 memory 를 게임의 난이도에 따라 큐의 size 를 변화시킨다. 난이도가 높을 수록 큐의 size 를 키워 더 많은 양을 기억할 수 있도록 했다.

3. 본론

3.1. system architecture

- 사용언어: C

게임의 전반적인 룰을 서론에서 다루었고 이제 어떻게 코드로 구현할 것인지 본론 부분에서 다루기로 한다.

첫째, 다빈치 코드 게임에서 플레이어들은 각각 자신이 뽑은 블록을 오름차순으로 배열하는데, 이 경우 컴퓨터 구현 내에서 자동으로 배열하도록 만든다. 이때, 각 사람이 가지고 있는 블록의 개수는 플레이어가 블록을 뽑을 때마다 변하고, 오름차순으로 정렬하기 위해서는 노드 간의 삽입이 자유로운 이중연결리스트가 적합하다 판단하여 이중연결리스트로 구현하였다.

둘째, '블록 더미'는 덱(deque)으로 구현했다. 게임 규칙에 의하면 플레이어는 블록 더미에서 검정 블록과 흰 블록 중에서 선택해서 뽑을 수 있다. 따라서 덱(deque)에 검정색 블록은 앞쪽(front)에, 흰색 블록은 뒤쪽(rear)에 배치하여 플레이어가 원하는 색의 블록에 따라 앞에서 혹은 뒤에서 delete 하여 반환하도록 했다. 검정색 블록을 담은 스택과 흰색 블록을 담은 스택 두 개를 이용할 수 있겠지만, '블록 더미' 라는 한 개념 아래에 덱이라는 한 자료구조개념을 사용했다.

셋째, 실제 Da Vinci code 보드게임은 2~4 인용의 게임이지만 핫 시트 방식의 진행은 게임 특성상 불가능하고, 유저들이 각자의 컴퓨터를 가지고 하는 게임을 만드는 것은 네트워크 통신을 요하기에 한정된 프로젝트 기간 내에 끝내지 못하리라 판단했다. 따라서 Bluffing 을 할 가능성이 있는 User(사람)를 상대로 유의미한 승률을 보이는 AI 를 구현하는 것을 최종 목표로 했다. 이에 따라 컴퓨터가 게임을 어떻게 받아들이고 결정을 내리는지에 대한 Decision making 프로세스가 추가되었다.

다빈치보드게임에서의 승리 전략들에 대한 AI decision making 구현과정을 정리해 보자면 다음과 같다.

유저의 공개된 블록을 기준으로 AI 가 숫자를 유추하기 위해서 블록의 후보군(candidates)을 등록하는 알고리즘을 필요로 한다.

유저가 지목했던 숫자의 블록은 유저에게 없을 가능성이 높기에, 유저가 지목했던 숫자를 기억하는 것은 컴퓨터가 게임을 이기는 것에 중요하게 작용한다. 따라서 유저가 부른 숫자를 기억하는 컴퓨터의 기억력을 큐(queue)로 구현했다.

하지만 유저를 100% 신뢰하여 유저가 지목할 숫자의 블록이 유저에게 없다고 판단해서 그 숫자를 후보군에서 제외한다면 컴퓨터의 승률이 확연히 떨어질 위험이 있다. 따라서 유저에 대한 신뢰도를 판단하는 함수가 필요하다. AI 가 적합한 guessing 과정을 통해 유저의 블록이 공개되었을 때, 그 블록의 숫자가 과거에 유저가 컴퓨터에게 추측한 숫자인지를 검사하고, 맞다면 추측할 당시 유저에게 그 카드가 있었는지 확인한다. 만약 과거에 유저가 AI 에게 물어봤던 숫자이면서 호명당시에 그 카드가 유저에게 있었다면 유저가 AI 에게 블러핑을 한 것이므로 신뢰도(reliability)를 감소시킨다.

3.2. Basic Algorithm Structure

1. 초기화 부분

1) 블록의 모임인 Deck, User 와 Com 가 소유하는 블록을 초기화한다.
2) 블록의 색깔과 플레이어의 순서를 결정해주는 전역변수를 초기화한다.

```
#include "DavinciCode.h"
#include "DoubleLinkedList.h"
#include "DoubleEndedQueue.h"
#include "ComputerGuess.h"
#include "gameUI.h"

Deque BlockDeck;    // 플레이어가 블럭을 뽑아올 블럭덱(Deck)
Block p_Block;    // USER가 소유하고 있는 블럭 head
Block c_Block;    // COM가 소유하고 있는 블럭 head

int num_white = BLOCK_LEN;
int num_black = BLOCK_LEN;
int TURN = COM;
int TURN_num = 0;
```

2. makeArrayRandom 함수

1) Deck 속의 블록 순서를 랜덤하게 만들기 위한 함수이다.
2) rand 함수를 이용해서 arr 배열의 값을 rs 배열에 임의로 저장하고 다시 arr 배열에 저장한다.

```
void makeArrayRandom(int * arr) {    // 매개변수로
    int len = BLOCK_LEN;
    int rs[BLOCK_LEN];
    int n = 0;

    while (len != 0) {
        int idx = rand() % len;
        rs[n++] = arr[idx];

        for (int i = idx; i < len - 1; i++)
            arr[i] = arr[i + 1];

        len--;
    }

    for (int i = 0; i < BLOCK_LEN; i++) {
        arr[i] = rs[i];
    }
}
```

3. InitDeck 함수

1) Deck 속의 값을 초기화하는 함수이다.

2) makeArrayRandom 함수를 이용해 배열의 값을 랜덤하게 만들고 malloc 함수를 이용해 BLOCK_LEN(사용하려는 블록개수) 만큼의 새로운 배열을 만들어 블록이 검은색일 경우 Deck 의 앞쪽에, 흰색일 경우 Deck 의 뒤쪽에 삽입한다.

```
41  void initDeck() {    // BlockDeck을 초기화
42      int arr[BLOCK_LEN] = { 0,1,2,3,4,5,6,7,8,9,10,11,JOKER };
43
44      for (int j = 0; j <= WHITE; j++) {
45          makeArrayRandom(arr);
46
47          for (int i = 0; i < BLOCK_LEN; i++) {
48              //printf("%d ", arr[i]);
49              Block * bnode = (Block*)malloc(sizeof(Block));
50              bnode->revealed = FALSE;
51              bnode->color = j;
52              bnode->number = arr[i];
53              bnode->rlink = NULL;
54              bnode->llink = NULL;
55
56              for (int k = 0; k < 26; k++)
57                  bnode->score[k] = 50;
58
59              if (j == 0)
60                  dq_addFront(&BlockDeck, bnode);
61              else
62                  dq_addRear(&BlockDeck, bnode);
63          }
64          printf("\n");
65      }
66  }
```

4. InsertAndSort 함수

1) Deck 에서 꺼낸 값을 User 혹은 Com 의 이중연결리스트에 다빈치코드식 정렬 순서에 맞게 삽입해주는 함수이다. 다빈치코드식 정렬이란, 기본적으로 숫자는 왼쪽에서 오른쪽으로 오름차순 정렬하고 숫자가 같은 경우 검은색이 흰색보다 앞 쪽에 오도록 한다.

2) 조커를 뽑은 경우 어디든 삽입할 수 있으므로 모든 위치에 삽입할 수 있으므로 모든 블록 사이의 위치를 출력하고 사용자가 선택한 곳에 삽입할 수 있도록 한다. (whereJoker 함수를 이용)

3) 컴퓨터가 뽑은 카드가 조커일 경우 랜덤하게 삽입하도록 한다.

```
void InsertAndSort(Block * PlayerBlock, Block * newBlock, int Player) {   // newBlock을 PlayerBlock리스트의 적절한 위치에 삽입
    Block * before;
    int n = -1; // JOKER가 있는 경우 넣을 위치 선택 변수
    int count;
    int x = LENGTH / 2 - (BLOCK_WIDTH / 2);
    int row = 17;

    if (Player == COM)
        add_to_known(newBlock);

    if (newBlock->number == JOKER) { // 넣으려는 블럭이 조커인 경우
        Block * p = PlayerBlock;
        int i = 0;

        if (Player == USER) {
            gotoP(50, 19);   printf("조커를 삽입할 위치를 고르세요");
            n = whereJoker(&p_Block);
        }
        else if (Player == COM) { // Computer의 경우
            for (p = PlayerBlock->rlink; p != PlayerBlock; p = p->rlink, i++);
            n = rand() % (i + 1);
        }

        p = PlayerBlock;
        for (i = 0; i < n; i++) {
            p = p->rlink;
        }

        before = p;
    }
```

4) 일반적인 카드의 경우 앞서 말한 다빈치코드식 정렬에 의해 삽입하는 것이 기본적이나, 만약 삽입하려는 위치의 앞이나 뒤에 조커가 존재할 경우 해당 조커의 앞 뒤까지도 선택해서 삽입할 수 있도록 사용자에게 선택권을 부여한다. 이 때, 조커가 연달아 두 개 있는 예외적인 상황이 발생할 수 있으므로 if 문을 이용해 먼저 처리해주도록 한다. (whereWithJoker 함수 이용)

5) Com 의 경우 선택할 수 있는 경우의 수 중 임의의 위치에 삽입하도록 한다.

```
    else {   // 넣으려는 블록이 조커가 아닌 경우

        n = -1; // JOKER가 있는 경우 넣을 위치 선택 변수 초기화
        before = dll_find_index(PlayerBlock, newBlock);      // 삽입할 위치 앞 노드
                                                             //printf("<%d>", before->block.number);
        clearChat();
        if (before->number == JOKER && before->rlink->number == JOKER) {  // 조커가 연달아 2개 있는 경우

            count = 2;

            if (Player == USER) {

                gotoP(x - 15, row);
                printf("방금 뽑은 블럭을 어디에 넣으시겠습니까?\n");
                printBlock(newBlock, x, row + 2);

                printBlock(before, x, row + 12);
                printBlock(before->rlink, x + 10, row + 11);
                n = whereWithJoker(count, x, row + 8);
            }
            else if (Player == COM) {
                n = rand() % 3;
            }
        }
```

6) 다음으로 삽입하려는 위치 앞에 Joker 가 한 개만 있는 경우에 예외처리해주는 코드

7) 마찬가지로 Com 의 경우 선택할 수 있는 경우의 수 중 임의로 선택하도록 한다.

```
123         else if (before->number == JOKER) {      // 삽입할 위치 앞에 JOKER가 1개 있는 경우
124
125             count = 1;
126
127             if (Player == USER) {
128
129                 gotoP(x - 15, row);
130                 printf("방금 뽑은 블럭을 어디에 넣으시겠습니까?\n");
131                 printBlock(newBlock, x, row + 2);
132
133                 printBlock(before, x, row + 11);
134                 n = whereWithJoker(count, x, row + 8);
135             }
136             else if (Player == COM) {
137                 n = rand() % 2;
138             }
139         }
140
141         if (n == 0) {
142             before = before->llink;
143         }
144         else if (n == 2) {
145             before = before->rlink;
146         }
147     }
148     dll_insert(before, newBlock);
149 }
```

5. choiceColor 함수

1) User 가 Deck 에서 새로운 블록이 뽑을 때 색을 결정하는 함수이다.

2) 먼저 dq_isEmpty 함수를 이용해 Deck 에 더 이상 블록이 없을 경우 무승부를 출력하게 된다.

3) Deck 에 블록이 존재할 경우 게임화면에 검은색과 흰색 블록이 몇 개 남았는 지 출력하고 사용자가 방향키를 이용해 선택할 수 있게 한다.

```
151  Block * choiceColor(Deque * pd) {   // 검정, 흰 블럭 선택 후 덱에서 블럭 반환하는 함수
152
153      int color = BLACK;
154      int select;
155
156      Block * newBlock = NULL;
157
158      if (dq_isEmpty(pd)) {
159          printf("더 이상 뽑아올 블럭이 없습니다.");
160          printf("무승부입니다.");
161          exit(1);
162      }
163      else
164      {
165          //while (getchar() != '\n');
166
167          int x = 58, y = 20;
168          int i;
169
170          setColor(10, 0);   gotoP(x - 1 - 1, y - 1);   printf("┌─────────┐");
171          gotoP(x - 1 + BLOCK_WIDTH / 2 - 3, y + 1); printf("남은블럭");
172          gotoP(x - 1 + BLOCK_WIDTH / 2, y + 3); printf("%d", num_black);
173          setColor(15, 0);   gotoP(x - 1 - 1 + 16, y - 1);   printf("┌─────────┐");
174          gotoP(x + 16 - 1 + BLOCK_WIDTH / 2 - 3, y + 1); printf("남은블럭");
175          gotoP(x + 16 - 1 + BLOCK_WIDTH / 2, y + 3); printf("%d", num_white);
176
177          for (i = 0; i < 5; i++) {
178
179              setColor(10, 0);
180              gotoP(x - 1 - 1, y + i);   printf("│");
181              gotoP(x + BLOCK_WIDTH, y + i);   printf("│");
182
183              setColor(15, 0);
184              gotoP(x - 1 - 1 + 16, y + i);   printf("│");
185              gotoP(x + BLOCK_WIDTH + 16, y + i);   printf("│");
186          }
187
188          setColor(10, 0);   gotoP(x - 1 - 1, y + 5);   printf("└─────────┘");
189          setColor(15, 0);   gotoP(x - 1 - 1 + 16, y + 5);   printf("└─────────┘");
190
191          resetColor();
192
193          gotoP(x - 1 - 1 + 4, y + 6); printf(" ▲ ");
194
195          gotoP(x - 1 - 1 - 9, y + 10); printf("방향키로 선택 후 Enter 또는 Space키를 누르세요.\n");
```

4) 왼쪽 방향키 값이 (0, 75)이고 오른쪽 방향키 값이 (0, 77)인 것을 이용해서 getch 함수를 두 번 이용해 왼쪽, 오른쪽 방향키를 인식하고 UI 로 화살표가 움직이도록 구현했다.

5) User 가 엔터나 스페이스바를 입력하게 되면 선택이 완료되고 while 문을 빠져나갑니다. 왼쪽일 경우 color 변수에 BLACK 값을, 오른쪽일경우 WHITE 값을 저장한다.

```
197         while (TRUE) {
198             select = getch();
199             if (select == ENTER || select == ' ')
200                 break;
201
202             if (select == 0 || select == 0xe0) {
203                 select = getch();
204                 if (select == LEFT) { // 왼쪽 방향키
205
206                     color = BLACK;
207
208                     gotoP(x - 1 - 1 + 20, y + 6); printf("    ");
209                     gotoP(x - 1 - 1 + 4, y + 6); printf(" ▲");
210                 }
211                 else if (select == RIGHT) { // 오른쪽 방향키
212
213                     color = WHITE;
214
215                     gotoP(x - 1 - 1 + 4, y + 6); printf("    ");
216                     gotoP(x - 1 - 1 + 20, y + 6); printf(" ▲");
217                 }
218             }
219         }
```

6) 선택된 color 값에 따라서 BLACK 일 경우 dq_deleteFront 함수를 이용해 Deck 의 앞쪽에서 블록을 꺼내 오고 WHITE 일 경우 dq_deleteRear 함수를 이용해 Deck 의 뒤쪽에서 블록을 꺼내 newBlock 변수에 저장한 뒤 newBlock 변수를 return 한다.

```
233         switch (color) {
234         case BLACK:
235             newBlock = dq_deleteFront(pd);
236             if (newBlock == NULL) {
237                 color = -1;
238                 break;
239             }
240             /*if (newBlock->number == JOKER) {
241             printf("뽑은 블럭 : 검은색 조커                                    \n");
242             printf("                                                            ", newBlock->number);
243             }
244             else {
245             printf("뽑은 블럭 : 검은색 %d                                      \n", newBlock->number);
246             printf("                                                            ", newBlock->number);
247             }*/
248
249             break;
250         case WHITE:
251             newBlock = dq_deleteRear(pd);
252             if (newBlock == NULL) {
253                 color = -1;
254                 break;
255             }
256             /*if (newBlock->number == JOKER){
257             printf("뽑은 블럭 : 흰색 조커                                      \n");
258             printf("                                                            ", newBlock->number);
259             }
260             else {
261             printf("뽑은 블럭 : 흰색 %d                                        \n");
262             printf("                                                            ", newBlock->number);
263             }*/
264
265             break;
266         }
267
268         gotoP(x - 1 - 1 - 9, y + 14);
269
270     }
271     return newBlock;
272 }
```

6. choiceColor_COM 함수

1) Com 가 Deck 에서 새로운 블록이 뽑을 때 색을 결정하는 함수이다.

2) 현재 Com 가 가지고 있는 블록의 상태를 파악해 상대가 쉽게 예측하지 못하도록 검정색이 많은 경우 흰색을 뽑고 흰색이 많은 경우 검정색을 뽑도록 하였다.

3) 선택된 color 값을 이용해 choiceColor 함수와 마찬가지로 Deck 에서 뽑은 블록을 newBlock 변수에 저장해 return 한다.

```
Block * choiceColor_COM(Deque * pd) {   // COM이 검정, 흰 블럭 선택 후 덱에서 블럭 반환하는 함수
    int color = -1;
    Block * newBlock = NULL;

    if (dq_isEmpty(&BlockDeck)) {
        printf("더 이상 뽑아올 블럭이 없습니다.");
        printf("무승부입니다.");
        exit(1);
    }
    else
    {
        while (color != BLACK && color != WHITE)
        {
            if (dq_isEmpty(pd))                 // 덱이 비었을때
                printf("블럭이 모두 소진되었습니다. 게임을 종료합니다.\n");
            else if (dq_isFrontEmpty(pd))       // 검정이 비었을때
                color = WHITE;
            else if (dq_isRearEmpty(pd))        // 하양이 비었을 때
                color = BLACK;
            else {
                Block* b;
                Block* tmp;
                tmp = &c_Block;

                int B_count = 0, W_count = 0;
                for (b = tmp->rlink; b != tmp; b = b->rlink) {
                    if (b->color == BLACK)
                        B_count++;
                    else if (b->color == WHITE)
                        W_count++;
                }
                // BLACK 많으면 WHITE 뽑고 WHITE 많으면 BLACK뽑는다
                // 같을땐 WHITE 뽑는다
                color = (B_count >= W_count ? WHITE : BLACK);
            }

            switch (color) {
            case BLACK:
                newBlock = dq_deleteFront(pd);
                if (newBlock == NULL) {
                    color = -1;
                    break;
                }
                break;
            case WHITE:
                newBlock = dq_deleteRear(pd);
                if (newBlock == NULL) {
                    color = -1;
                    break;
                }
                break;
            }
        }
    }
    return newBlock;
}
```

7. currentDeck 함수, showBlocks 함수 및 게임 진행 전역변수 초기화

1) currentDeck 함수는 현재 Deck 에 남은 검정색과 흰색 블록의 개수를 표시해준다.

2) showBlocks 함수는 화면을 초기화하고 게임 테두리와 현재 Com, User 가 가지고 있는 블록을 화면에 표시해준다.

3) replay 변수는 User 가 Com 의 블록을 맞췄을 경우 계속할 지 여부를 결정한다. True 일 경우 계속, False 일 경우 멈춘다. replay_COM 변수의 경우 Com 이 User 의 블록을 맞췄을 경우에 해당한다.

```
332  void currentDeck() {   // 현재 블럭 DECK에 남아있는 흰색, 검은색 블럭의 개수를 출력
333      printf("\n%d개의 검은 블럭과 %d개의 흰 블럭이 남아있습니다.\n\n", num_black, num_white);
334  }
335  void showBlocks() {
336      CLEAR();
337      showEdge();
338      showChat();
339
340      printHand(&c_Block);
341      printHand(&p_Block);
342  }
343
344  char replay = FALSE;     // 유저가 맞췄을 때 게임을 계속 진행할 지
345  char replay_COM = FALSE;// 컴퓨터가 맞췄을 때 게임을 계속 진행할 지
346  Block * newBlock = NULL;
```

8. isEndUser 함수, isEndCom 함수

1) isEndUser 함수는 현재 User 가 가지고 있는 블록이 전부 공개되었다면 TRUE 를, 전부 공개되지는 않았다면 FALSE 를 return 해 TRUE 일 경우 Com 이 이겼음을 알려준다.

2) isEndCom 함수의 경우 반대로 Com 이 가지고 있는 블록이 전부 공개되었는지 알려준다.

```
503  int isEndUser(Block * p_Block) {   // 게임이 끝났는지 판별
504      Block * p;
505
506      for (p = p_Block->rlink; p != p_Block; p = p->rlink) {
507          if (p->revealed == FALSE)
508              return FALSE;
509      }
510      return TRUE;
511  }
512  int isEndCom(Block * c_Block) {
513      Block * p;
514      for (p = c_Block->rlink; p != c_Block; p = p->rlink) {
515          if (p->revealed == FALSE) {
516              return FALSE;
517          }
518      }
519      return TRUE;
520  }
```

9. play 함수

1) 실제 게임을 진행하는 역할을 하는 함수이다.

2) 처음에 showBlocks, showChat 함수를 이용해 현재 가지고 있는 블록과 게임을 진행하는 메인화면을 콘솔에 출력한다.

3) TURN 변수를 이용해 User 의 차례인지 Com 의 차례인지 결정해준다.

4) 전역변수 replay 변수가 TRUE 일 경우 새로 블록을 뽑지 않고 추측해야하므로 replay 변수가 FALSE 인 경우에만 새로운 블록을 가져올 수 있도록 if 문으로 처리했다.

5) 앞서 정의했던 choiceColor 함수와 InsertAndSort 함수를 이용해 블록을 선택, 정렬하고 화면에 출력한다.

6) do~while 문을 이용해 User 가 Com 의 공개되지 않은 블록을 선택할 때까지 반복한다.

```
348  void play() {
349      if (TURN == USER) {
350
351          showBlocks();
352          showChat();
353          gotoP(LENGTH / 2 - 10, 13); printf("당신의 차례입니다.");
354          if (replay == FALSE) {
355              gotoP(LENGTH / 2 - 25, 16);  printf("새로운 블럭을 가져옵니다.");
356              newBlock = choiceColor(&BlockDeck);
357              TURN_num++;
358              init_score(newBlock, &p_Block);
359              InsertAndSort(&p_Block, newBlock, USER);
360              printHand(&p_Block);
361
362          }
363          gotoP(LENGTH / 2 - 25, 16);  printf("                          ");   // "새로운 블럭을 가져옵니다." 지우기
364          // 박싱 테스트 BlockBoxing(LENGTH / 2 - (BLOCK_WIDTH + 2) * (4 / 2), 3 + 3);
365
366
367          Block * selected_Block;      // USER가 guess할 블럭
368          int choiceNumber = -1;  // USER가 guess한 수
369
370
371          do {
372              clearChat();
373              selected_Block = selectBlock(&c_Block);
374
375              if (selected_Block->revealed == TRUE) {
376                  gotoP(LENGTH / 2 - 30, 18);  printf("이미 공개된 블럭입니다. 다른 블럭을 추측하세요.");
377                  gotoP(LENGTH / 2 - 30, 19); printf("(계속하려면 엔터)");
378                  getchar();
379              }
380              else
381                  break;
382
383          } while (TRUE);
384
385          clearChat();
386          choiceNumber = choose_guessNumber();
387
388          insert_USER_guess(2 * choiceNumber + selected_Block->color);
```

7) 처음 if 문에서는 선택한 블록의 숫자와 추측한 숫자가 일치할 경우 맞춘 블록을 공개하고 계속 추측할 지 여부를 결정하게 된다.

8) 계속하지 않는다고 결정하면 턴이 끝나고 TURN 변수에 COM 이 저장되어 차례가 넘어간다.

9) 계속하기로 결정하면 replay 변수가 TRUE 가 되고 새로운 블록을 뽑지 않고 계속해서 추측을 하게 된다.

10) else 문에서 추측한 블록의 숫자가 틀렸을 경우 방금 뽑아온 블록(newBlock)을 공개하고 TURN 변수에 COM 이 저장되어 차례가 넘어간다.

```
390         if (selected_Block->number == choiceNumber) {
391             selected_Block->revealed = TRUE;
392             while (TRUE) {
393                 clearChat();
394 
395                 gotoP(LENGTH / 2 - 25, 16); printf("맞췄습니다. 계속하시겠습니까? [Y/N] : ");
396                 scanf("%c", &replay);
397                 while (getchar() != '\n');
398 
399                 if (replay == 'N' || replay == 'n') {
400                     gotoP(LENGTH / 2 - 25, 18); printf("계속하지 않겠다고 결정.\n");
401                     replay = FALSE;
402                     TURN = COM;
403                     gotoP(LENGTH / 2 - 25, 20); printf("한 턴이 끝났습니다.(계속하려면 엔터)\n");
404                     getchar();
405                     return;
406                 }
407 
408                 else if (replay == 'Y' || replay == 'y') {
409                     gotoP(LENGTH / 2 - 25, 18); printf("계속하겠다고 결정.\n");
410                     replay = TRUE;
411                     TURN = USER;
412                     gotoP(LENGTH / 2 - 25, 20); printf("한 턴이 끝났습니다. (계속하려면 엔터)\n");
413                     getchar();
414                     return;
415                 }
416             }
417         }
418 
419 
420         else {
421             gotoP(LENGTH / 2 - 25, 16); printf("틀렸습니다. 이번 차례에 가져온 블럭을 공개합니다.");
422             gotoP(LENGTH / 2 - 25, 17); printf("(계속하려면 엔터)");
423             newBlock->revealed = TRUE;
424             add_to_known(newBlock);
425             TURN = COM;
426             replay = FALSE;
427             getchar();
428             return;
429         }
430     }
```

11) 다음은 TURN 변수에 COM 이 저장되어 Com 의 차례일 경우를 나타낸다.

12) 앞서 정의한 choiceColor_COM 함수를 이용해 새로운 블록을 뽑고 InserAndSort 함수를 이용해 이중연결리스트에 삽입, 정렬한다.

13) Sleep 함수를 이용해 Com 가 추측을 하고 있다는 것을 콘솔에 출력해준다.

```
    else if (TURN == COM) {

        clearChat();
        showBlocks();
        gotoP(LENGTH / 2 - 10, 13); printf("컴퓨터의 차례입니다. ");


        if (replay_COM == FALSE) {      // 컴퓨터의 첫 차례인 경우 draw
            gotoP(LENGTH / 2 - 25, 16); printf("새로운 블럭을 가져옵니다.\n");
            // 컴퓨터가 draw하는 함수.
            newBlock = choiceColor_COM(&BlockDeck);
            newBlock->get_turn = ++TURN_num;
            InsertAndSort(&c_Block, newBlock, COM);
        }


        gotoP(LENGTH / 2 - 25, 18); printf("Computer is thinking");
        for (int i = 0; i < 6; i++) {
            if (i < 3) {
                printf(".");
                Sleep(200);
            }
            else {
                printf("\b \b");
                Sleep(200);
            }
        }
        printf("..DONE!");
```

14) guess 함수를 이용해 Com 가 추측하게 되고 추측한 결과를 화면에 출력해준다.

15) Com 이 추측에 성공했을 경우 User 와 마찬가지로 계속할지 말지 여부를 결정해 계속하기로 결정했다면 replay_COM 변수에 TRUE 를 저장하고 새로운 추측을 하게 되고, 턴을 마치기로 결정했다면 replay_COM 변수에 FALSE 가 저장되고 TURN 변수에 USER 가 저장되어 차례를 넘긴다.

16) User 와 마찬가지로 Com 가 추측에 실패했을 경우, 방금 뽑아온 블록(newBlock)을 공개하고 TURN 변수에 USER 를 저장해 차례를 넘긴다.

```
        int choicePosition = rs[0];
        int choiceNumber = rs[1] / 2;
        int computerWantsToGo = rs[2];

        Block * p = dll_getNodeAt(&p_Block, choicePosition);

        if (p->number == choiceNumber) {
            p->revealed = TRUE;
            gotoP(LENGTH / 2 - 31, 20); printf("컴퓨터가 당신의 %d번째 블럭의 숫자를 맞췄습니다.\n", choicePosition);
            add_to_known(p);

            if (!computerWantsToGo) {
                gotoP(LENGTH / 2 - 25, 21); printf("컴퓨터가 차례를 당신에게 넘겼습니다.\n");
                gotoP(LENGTH / 2 - 25, 23); printf("한 턴이 끝났습니다.(계속하려면 엔터)\n");
                getchar();
                TURN = USER;
                replay_COM = FALSE;
            }

            else {
                gotoP(LENGTH / 2 - 25, 21); printf("컴퓨터가 계속 플레이 하겠다고 결정했습니다.\n");
                gotoP(LENGTH / 2 - 25, 23); printf("한 턴이 끝났습니다.(계속하려면 엔터)\n");
                getchar();
                TURN = COM;
                replay_COM = TRUE;
            }
        }

        else {
            gotoP(LENGTH / 2 - 31, 20); printf("컴퓨터가 당신의 %d번째 블럭이 \"%d\" 이라고 예상했으나, 틀렸습니다.\n", choicePosition, choiceNumber);
            gotoP(LENGTH / 2 - 25, 22); printf("한 턴이 끝났습니다.(계속하려면 엔터)\n");
            newBlock->revealed = TRUE;
            getchar();
            TURN = USER;
            replay_COM = FALSE;
        }
```

10. main 함수

1) showEdge, showMain 함수를 이용해 콘솔에 게임화면을 출력한다.

2) dq_init, initDeck, dll_init 함수를 이용해 Deck 과 User, Com 의 블록 이중연결리스트를 초기화한다.

3) rand 함수를 이용해서 Com 에게 처음 블록을 임의로 나눠준다.

4) User 의 경우 choiceColor 함수와 InsertAndSort 함수를 이용해 처음 시작 블록의 색을 선택하고 이중연결리스트에 삽입, 정렬한다.

5) printHand 함수를 이용해 콘솔창에 현재 가지고 있는 블록의 색과 순서를 출력한다.

```
522  int main() {
523      srand((int)time(NULL));
524
525      SetCursorInvisible();
526      system("title Da Vinci Code");
527      system("mode con: cols=140 lines=51");
528      showEdge();
529      showMain();
530      init(&USER_guess);
531
532      dq_init(&BlockDeck);
533      initDeck();
534      dll_init(&p_Block);
535      dll_init(&c_Block);
536
537      Sleep(2000);
538
539      CLEAR();
540      showEdge();
541      showChat();
542
543      for (int i = 0; i < INIT_BLOCK; i++) {   // 처음 패 나눠주기(COM)
544          int n = rand() % 2;
545          if (n == BLACK)
546              InsertAndSort(&c_Block, dq_deleteFront(&BlockDeck), COM);
547          else if (n == WHITE)
548              InsertAndSort(&c_Block, dq_deleteRear(&BlockDeck), COM);
549      }
550      for (int i = 0; i < INIT_BLOCK; i++) {  // 처음 패 나눠주기(USER)
551          clearChat();
552          gotoP(LENGTH / 2 - 5, 14);  printf("초기 드로우");
553          InsertAndSort(&p_Block, choiceColor(&BlockDeck), USER);
554          printHand(&p_Block);
555      }
556      printHand(&c_Block);
557      // 처음 패 나눠주기 완료
```

6) while 문을 이용해 User 혹은 Com 의 블록이 모두 공개될 때까지 play 함수를 반복한다. (isEndUser, isEndCom 함수 이용)

7) User 혹은 Com 의 블록이 모두 공개되지 않았지만 만약 Deck 의 모든 블록을 가져왔을 경우 무승부로 처리하고 게임을 종료한다. (dq_isEmpty 함수 이용)

8) isEndUser 함수를 이용해 User 의 블록이 모두 공개되었을 경우 Com 이 이겼음을 출력하고, isEndCom 함수를 이용해 Com 의 블록이 모두 공개되었을 경우 User 가 이겼음을 출력한 뒤 게임을 종료한다.

```
559     while (!isEndUser(&p_Block) && !isEndCom(&c_Block)) {
560         play();
561         if (dq_isEmpty(&BlockDeck)) {
562             printf("더 이상 뽑아올 블럭이 없습니다.");
563             printf("무승부입니다.");
564             exit(1);
565         }
566
567     }
568
569     if (isEndUser(&p_Block) && isEndCom(&c_Block)) {
570         printf("무승부입니다.");
571         exit(1);
572     }
573     else if (isEndUser(&p_Block)) {
574         gotoP(LENGTH / 2 - 25, 25);
575         printf("컴퓨터가 승리하였습니다.");
576         exit(1);
577     }
578     else if (isEndCom(&c_Block)) {
579         gotoP(LENGTH / 2 - 25, 25);
580         printf("당신이 승리하였습니다.");
581         exit(1);
582     }
583
584     Sleep(5000);
585 }
```

3.3. Computer Guessing Algorithm

```
int *  rs = guess(&p_Block);

gotoP(LENGTH / 2 - 25, 20); printf("게스 결과: %d %d %d₩n", rs[0], rs[1], rs[2]);

int choicePosition = rs[0];
int choiceNumber = rs[1] / 2;
int computerWantsToGo = rs[2];
```

COM 이 USER 의 패에서 한 블록을 선택해 추측하는 부분은 함수 guess()로 구현했다. 위는 play()함수에서 guess 를 호출하여 사용하는 부분이다.

초기에는 다빈치코드를 2P 에 한정 짓지 않고 4P 까지 확장할 계획이었다. 따라서 누구의 패를 대상으로 추측을 할 지 특정하기 위해 guess()는 목표 상대방의 블록 헤드 포인터를 입력으로 받는다.

상대방의 블록 헤드 포인터를 시점으로, COM 은 상대방 패의 모든 블록에 대해 가중치를 갱신하고, 그 중 가장 큰 가중치를 같는 값과 그 값이 속한 블록의 position 을 반환한다. 또한, 한번의 추측이 끝난 후 또 한번 추측할 것인지 결정하기 위해 나머지 블록의 가중치를 다시한번 검사한다. 만약 검사 결과 계속해서 추측하는 것이 유리하다 판단되면 computerWantsToGo 에 1 이 할당된다.

```
int* guess(Block* OpponentBlock_head) {              // COM의 차례에 드로우 이후 호출
                                                     // input: 상대방의 블럭 head

    weight_init(OpponentBlock_head);                 // initial candidates
    weight_add(OpponentBlock_head);                  // reliability를 고려한 가중치를 더해주는 함수
                                                     // int max[3] = guess_getMax(OpponentBlock_head);   // [position][guess_number][score]
    int* max;                                        // [position][guess_number][score]
    max = weight_getMax(OpponentBlock_head);
    int guess1[2] = { max[0] ,max[1] };              // [position][guess_number]
    after_guess(OpponentBlock_head, guess1);         // guess를 했으니 그 블럭에서 그 경우의 수를 없애주는 함수

    // 계속해서 guess 할지 결정
    int * result = (int*)malloc(sizeof(int) * 3);
    result[0] = max[0];
    result[1] = max[1];
    result[2] = 0;                                   // [position][guess_number][repeat?]

    int* max2;                                       // [position][guess_number][score]
    max2 = weight_getMax(OpponentBlock_head);
    if (max2[2] >= LEAST_SCORE_TO_GUESS) {           // 그 다음 큰 가중치값도 LEAST_SCORE_TO_GUESS 이상일 경우만
        Block* current_block;
        int i = 0;
        for (current_block = OpponentBlock_head->rlink; current_block != OpponentBlock_head; current_block = current_block->rlink, i++)
            if (i == result[1]) break;

        if ((2 * current_block->number + current_block->color) == result[1])   // 방금 COM guess가 맞았을 경우만
            result[2] = 1;                           // 이 경우 만족할 경우만 더 해
    }

    return result;
}
```

함수 guess()는 아래와 같다.

guess()는 기본적으로 상대방 패 블록의 가중치를 갱신해주는 두 함수(weight_init(), weight_add())와 계속해서 guess 할 것인지 결정하는 부분으로 구성되어 있다.

weight_init()과 weight_add()는 guess()와 마찬가지로 목표 상대방의 블록 헤드를 인자로 받는다.

```
void weight_init(Block * head) {        // input: 상대방(USER)의 블럭 head

    for (Block* b = head->rlink; b != head; b = b->rlink) {
        if (b->revealed == 0) {       // 블럭이 비공개 상태면-
            for (int i = 0; i < 26; i++)       // 그 블럭의 가중치배열의 이미 알려진 놈들의 가중치를 0으로 만들어준다
                if (known_block[i] == 1)
                    b->score[i] = 0;

            // 다른 색깔 경우의 수 0으로 만들어주기
            if (b->color == BLACK) {       // 홀수 0 만들어주기
                for (int i = 1; i < 26; i += 2)
                    b->score[i] = 0;
            }
            else {
                for (int i = 0; i < 26; i += 2)       // 짝수 0 만들어주기
                    b->score[i] = 0;
            }
        }
    }
    // 이제 USER 패의 공개된 블럭들을 기준으로 불가능한 경우의 수의 가중치를 0으로 만들어준다.
    for (Block* b = head->rlink; b != head; b = b->rlink)
    {
        // b가 공개된 블럭이라면 && 조커가 아니라면
        if (b->revealed == 1 && b->number != 12)
        {
            // 앞에있는 블럭들의 가중치값 정리
            for (Block* front = head->rlink; front != b; front = front->rlink)
                // b 앞의 '자신보다 큰 경우의 수' 가중치 0만들기
                for (int i = 2 * b->number + b->color; i < 24; i++)   // i=24,25은 조커자리이므로 건드리지 않습니다
                    front->score[i] = 0;

            // 뒤에있는 블럭들의 가중치값 정리
            for (Block* back = b->rlink; back != head; back = back->rlink)
                // b 뒤의 '자신보다 작은 경우의 수' 가중치 0만들기
                for (int i = 0; i <= 2 * b->number + b->color; i++)
                    back->score[i] = 0;
        }
    }
}
```

weight_init()은 그 블록에 존재할 가능성이 없는 모든 경우의 수에 대한 가중치를 0 으로 초기화해준다.

우선, 각 블록은 유일하므로 공개된 블록과 같은 경우의 수의 가중치를 0 으로 만든다.

또한 다른 색깔에 해당하는 경우의 수에 대한 가중치 값도 0 으로 만들어준다.

그 다음, 다빈치코드에서 각 패는 오름차순으로 정렬되어야 하므로, 상대의 공개된 블록들을 지표로 불가능한 경우의 수에 대한 가중치를 0 으로 만든다.

```
void weight_add(Block* OpponentBlock_head) {

    // 공개 상태의 블럭 current_block, current_block2 사이의 count개의 블럭들의 가중치 더해줘
    for (Block* current_block = OpponentBlock_head->rlink; current_block != OpponentBlock_head; current_block = current_block->rlink)
    {
        if (current_block->revealed == 0)
        {
            int count = 0;
            for (int i = 0; i < 26; i++)
                if (current_block->score[i] != 0) count++;      // final candidatesd들이 가중치를 나눠갖도록 하려면 우선 몇개인지 알아야

            for (int i = 0; i < 26; i++)
                if (current_block->score[i] != 0)               // 가중치가 0이 아니면~
                {
                    int is_it_recent_guess = 0;      // 1: 방금 turn USER의 guess이다.
                                                     // 0: 아니다.

                    int index = USER_guess.rear;
                    do {
                        if (USER_guess.queue[index].guess_number == current_block->number) {
                            is_it_recent_guess = 1;
                        }
                        index--;
                    } while (USER_guess.queue[index].guess_turn == USER_guess.queue[USER_guess.rear].guess_turn);
                    // 방금전 턴 guess 전부 검사
                    if (is_it_recent_guess)          // USER가 방금 guess한 number면
                        current_block->score[i] += (100 - user_reliability) / count;
                    else                             // 아니면
                        current_block->score[i] += (user_reliability) / count;

                }
        }
    }
}
```

weight_add()는 그 블록에 존재할 가능성이 있는 숫자들에 가중치를 더해주는 함수이다.
weight_init()에서 불가능한 경우의 수를 모두 제거해 주었다면, weight_add()는 남아있는 경우의 수 각각에 상응하는 가중치 값을 더해준다. 한 블록에 총 더해지는 가중치 값은 user_reliability 에 따라 달라진다. 또한 경우의 수가 많은 블록일수록 가중치 값을 더 나눠 갖기 때문에 각 블록에 더해지는 가중치 값은 낮아진다.
이 두 함수가 호출되고 나면, COM 이 가중치를 갱신하는 작업은 완료된다. 그 다음 해야 할 일은, COM 이 계속해서 추측을 할 것인지 결정하는 것이다. 다빈치코드에서는 매 턴 블록을 드로우한 다음 상대방 패에서 한 블록을 골라 그 수를 추측하고, 만약 맞았을 경우에는 이어서 추측을 할 것인지 결정할 수 있다. 만약 계속해서 추측을 할 경우에는 추측이 틀려서 이번 턴 드로우한 블록을 공개해야 하는 위험을 감수해야 한다. 하지만 게임 후반에는 공개된 블록이 많아지면서 경우의 수가 매우 줄어들게 되기 때문에 추측을 이어가는 것이 현명한 판단이 될 수도 있다.

```
    int* max2;                                  // [position][guess_number][score]
    max2 = weight_getMax(OpponentBlock_head);
    if (max2[2] >= LEAST_SCORE_TO_GUESS) {      // 그 다음 큰 가중치값도 LEAST_SCORE_TO_GUESS 이상일 경우만
        Block* current_block;
        int i = 0;
        for (current_block = OpponentBlock_head->rlink; current_block != OpponentBlock_head; current_block = current_block->rlink, i++)
            if (i == result[1]) break;

        if ((2 * current_block->number + current_block->color) == result[1])   // 방금 COM guess가 맞았을 경우만
            result[2] = 1;                      // 이 경우 만족할 경우만 더 해
    }

    return result;
```

위 사진은 guess() 내부 COM 이 계속해서 추측을 할지 결정하는 부분이다.
weight_getMax()로 가장 큰 가중치 값을 반환한 뒤 그것이 특정 값 이상이면 1 을 반환한다. 이 특정 값은 몇 번의 테스팅을 걸쳐서 결정했다.
guess() 외에도 COM 의 추측 알고리즘에 필요한 몇가지 부수적인 함수가 있다. init_score()과 did_USER_lie()이다.

아래 사진은 init_score() 함수의 스크린샷이다.

```
void init_score(Block* b, Block* OpponentBlock_head) {  // USER가 새로운 블록을 뽑았을 때 호출
                                                        // input: 새로 뽑은 블럭
                                                        // USER이 draw한 블럭의 점수는 기존 블럭들의 점수의 평균
    int score = 0;
    int count = 0;
    for (Block* current_block = OpponentBlock_head->rlink; current_block != OpponentBlock_head; current_block = current_block->rlink) {
        for (int i = 0; i < 26; i++) {
            if (current_block->score[i] != 0) {
                score += current_block->score[i];
                count++;
            }
        }
    }
    for (int i = 0; i < 26; i++)
        b->score[i] = score / count;                    // int여서 소숫점 씹히겠지만 그정도는 괜찮겟지?
    b->get_turn = TURN_num;
}
```

초기 덱 구성에서는 가중치를 50 으로 맞춰 놨기 때문에 상대방이 새로운 블록을 드로우했을 경우 그 블록의 가중치 값들을 갱신해 주어야 한다. 만약 갱신해주지 않으면 새로운 블록의 가중치값들은 다른 블록과 비교해 너무 낮아서 weight_add()를 통해 가중치를 더해주어도 COM 이 추측을 하기에 매우 후순위일 것이다. 새로운 블록의 가중치 값들은 기존 블록 전부의 가중치 값의 평균이다.
아래 사진은 did_USER_lie() 함수의 스크린샷이다.

```
void did_USER_lie(Block* revealed) {                    // COM이 USER의 패를 맞춰서 앞면이 보일 때 호출해서 검사
                                                        // USER이 guess했는데 틀려서 그 턴에 드로우한 블럭이 공개됐을 때도 호출
                                                        // input: 이번에 맞춰서 공개된 블럭
    int index = USER_guess.rear;
    do {
        if (USER_guess.queue[index].guess_number == revealed->number) {     // 만약 공개된 블럭의 number가 USER이 guess한 number중에 있다면
            if (revealed->get_turn <= USER_guess.queue[index].guess_turn)   // 만약 공개된 블럭을 드로우한 이후에 guess를 한거라면
                user_reliability--;                                         // 어느정도 더하고 뺄지는 조정.
            else
                user_reliability++;
        }
        index--;
    } while (USER_guess.queue[index].guess_turn != USER_guess.queue[USER_guess.rear].guess_turn);
    // 방금전 턴 guess 전부 검사
    // 만약 공개된 블럭의 number가 USER이 guess한 number중에 없다면, 아무것도 안함
}
```

상대방의 패가 공개될 때마다 COM 은 과거에 상대방이 블러핑 한적이 있는지 검사하고, 만약 있다면 신뢰도를 감한다.

4.결론

4.1. 개선점 및 향후 발전 방향

GUI 구현 경험이 있는 조원이 없었기에 게임 진행화면을 콘솔 창에서 구현했다. 콘솔 창에서 구현하다 보니 화면의 자리가 부족해서 2 인용에 머물렀다. GUI 라이브러리를 이용한다면 디자인의 완성도를 높임과 동시에 4 인용 플레이까지 구현할 수 있을 것이다.

또한, 네트워크 통신으로 온라인 플레이가 가능하게끔 구상하였으나 시간상의 이유로 구현하지 못했다. 기존에 있던 'Enigma' 게임과 비교하였을 때, 유일하게 비교되는 단점이었다고 볼 수 있다.

다빈치 코드 보드게임은 한판에 15 분 남짓의 시간이 소요된다. 컴퓨터 프로그램으로 구현된 다빈치 코드 게임은 이것보다는 짧지만 여전히 5 분 정도의 시간이 소요된다. 따라서 한번 디버깅 하기 위해서는 프로그램을 실행하는 데만 5 분이 걸린다. 따라서 매우 많은 시간을 투자했음에도 아직 컴퓨터의 의사결정에 필요한 몇 가지 상수 값들이 미흡하다. 만약 더 많은 시연을 거쳐 이 값들을 최적화 시킨다면 컴퓨터의 승률은 눈에 띄게 향상되리라 판단한다.

4.2. 역할 분담

-Da Vinci code game 전체적인 개괄 코드 구성: 김윤성, 장연석
-Decision making Algorithm for computer 코드 구현: 정호용
-중간, 기말 보고서, 발표: 김은채