|  |
| --- |
|  |
| **기초공학설계(3반) 2017학년도 1학기 프로젝트 (2048 게임)** |
|  |
|  |
|  |
| 컴퓨터공학과 20161620 이수연 |

**<2048게임 프로젝트 보고서>**

**전공: 컴퓨터공학과 학년: 2학년 학번: 20161620 이름: 이수연**

**1. 설계 문제 및 목표**

이탈리아의 웹 개발자 가브리엘레 치룰리 (Gabriele Cirulli)가 만든 퍼즐 게임인 2048 게임을 구현한다. 2048 게임의 실행 방법은 다음과 같다.

시작할 때 4 x 4 판 위에 임의의 두 위치에 각각 2(또는 4) 블록이 있는데 사용자는 상하좌우 입력 키를 누를 수 있으며, 사용자가 입력 키를 누를 때마다 과정 1과 2를 진행한다.

**과정 1)** 사용자가 누른 입력 키 방향으로 모든 블록들을 한 칸씩 반복적으로 이동할 수 있을 때까지 이동시킨다. 블록들은 자신이 이동하고자 하는 칸에 블록이 존재하거나 판 밖으로 벗어나게 될 경우가 아니면 계속 사용자가 이동하는 방향으로 이동한다.

예외적으로, 블록 A가 이동하고자 하는 칸에 있는 블록을 블록 B로 정의 할 때 블록 A와 블록 B에 표시 된 숫자가 동일 할 때, 블록 B 위치에 두 블록이 하나의 블록으로 뭉쳐진다. 이 때, 뭉쳐진 블록에 표시 된 숫자는 블록 A, B에 표시 된 숫자의 합이다. 하나로 뭉쳐진 블록도 사용자가 누른 방향으로 한 칸씩 이동 할 수 있을 때까지 이동시킨다. 단, 한 번 뭉친 블록은 과정 2를 진행하기 전까지 다른 블록과 다시 뭉쳐질 수 없다.

**과정 2)** 모든 블록들이 사용자가 입력한 방향으로 더 이상 이동 시키는 것이 불가능하다면, 4 x 4 판 중 블록이 없는 칸에 새로운 2(또는 4)의 블록을 생성한다. 단 모든 블록들이 사용자가 입력한 방향으로 더 이상 이동 시키는 것이 불가능하면서 과정 1을 진행하면서 위치가 변하거나 새로 뭉쳐진 블록이 없으면, 새로운 블록을 생성하지 않는다.

이를 반복해서 2로부터 2048을 만들면 게임이 끝나게 된다. 만약 2048을 만들기 전 이동할 수 없는 경우, 즉 16칸이 꽉 채워져 있으면서 인접한 두 칸이 같지 않을 때 게임 오버가 되며, 2048을 만듦과 동시에 이동 불가이면 역시 게임 오버가 된다.

하지만 최근의 변형 된 2048 Game의 경우 2048을 만드는 것이 목표가 아닌 4096, 9192…와 같이 더 큰 수를 만드는 것이 목표인 것도 있다.

이번 프로젝트에서는 위에서 설명한 변형된 2048 게임을 구현하는 것을 목표로 한다.

**기초공학설계(3반) 20161620 이수연**

**2. 프로그램 실행 흐름도 및 설명**

init

**2-1. Flow Chart**

Return 값이 1이 아닐 때 (game over가 아닐 때) 루프

Return 값이 1이 아닐 때

Return 값이 -1일 때 (정상 키 X)

게임이 끝날 때까지 방향키에 따라 판을 새로 그리고, 랜덤으로 숫자를 생성하는 루프

초기 화면 그리기 (랜덤 숫자 2개)

print game over

(END)

is\_game\_over

draw\_board

make\_two\_or\_four

set\_board

Getcommand

draw\_board

make\_two\_or\_four

make\_two\_or\_four

**2-2. 함수 설명**

**(0). 전역 변수**

\* int game\_over : 게임 오버 flag

\* int b[4][4] : 판을 그릴 4\*4 배열

\* FILE \*fp : output.txt 파일에 쓰기 위해 선언하는 파일 포인터

**(1). void init() : 배열 b[][] 값을 모두 0 으로 초기화 시켜주는 함수**

void init(){

int i, j;

game\_over = 0;

for(i=0; i<4; i++)

{

for(j=0; j<4; j++)

b[i][j] = 0;

}

}

**(2). int make\_two\_or\_four() : 2를 2/3의 확률로, 4를 1/3의 확률로 랜덤으로 수를 생성하는 함수 , 만약 랜덤으로 지정할 위치가 없으면 (판이 모두 채워져 있으면) 0을 전달하고 아니라면 랜덤으로 생성한 숫자를 전달한다.**

int make\_two\_or\_four(){

int num,locate,is\_zero=0;

int arr[3]={2,2,4};//2를 2/3의 확률로, 4를 1/3의 확률로 생성하기 위해 배열 설정

num=arr[rand()%3];//랜덤으로 수 생성

while(1)

{

locate=rand()%16;//배열 16곳 중에 랜덤으로 위치지정

if(b[locate/4][locate%4]==0)//그 위치가 0이면

{

is\_zero=1;//flag설정하고 break

break;

}

else//그 위치가 0이 아니라 채워져 있으면 다시 랜덤 위치 생성, flag는 0

is\_zero=0;

}

if(is\_zero==1)//flag가 1이면 0인 곳이 있으므로 생성한 랜덤숫자 전달

return num;

else//flag가 0이면 생성할 위치가 없다는 의미이므로 0 전달

return 0;

/\* user code \*/

/\* return 2 or 4 that makes in this times \*/

/\* if can not make two or four, then return 0 \*/

}

**(3). void draw\_board (int tot, int command) : command(방향키)를 입력 받아 필드를 새로 그리고 점수를 계산한다. 방향키, 점수, 판의 모양을 판을 새로 그릴 때마다 파일에 써준다.**

void draw\_board(int tot, int command){

int i, j, k, c[8][8], score;

/\* console clear \*/

system("clear");

score = 0;

/\* user code \*/

for(i=0; i<4; i++)//배열에서 기존보다 큰 수가 나올 때마다 score에 저장

{

for(j=0; j<4; j++)

{

if(b[i][j]>score)

score=b[i][j];

}

}

/\* calculate score & check sum of all block equasls variable tot \*/

//파일에 입력, 필드 생성(기존 코드)

printf(" Score : %d\n", score);

fprintf(fp, "%d %d\n", score, command);

for(i=0; i<4; i++){

for(j=0; j<4; j++){

c[i][j] = 32;

for(k=0; k<50; k++){

if( b[i][j] == (1<<k) ){

c[i][j] = 32 + (k%6);

break;

}

}

}

}

for(i=0; i<4; i++){

for(j=0; j<4; j++)

fprintf(fp, "%d ", b[i][j]);

fprintf(fp, "\n");

}

for(i=0; i<4; i++){

puts("");

puts("");

for(j=0; j<4; j++)

printf("\033[%dm%5d\033[0m", c[i][j], b[i][j]);

puts("");

puts("");

}

}

**(4). int getch(void) : 사용자로부터 입력 받은 키를 반환해준다.**

(기존에 제공되었던 코드에서 수정하지 않음)

int getch(void)

{

char ch;

int error;

static struct termios Otty, Ntty;

fflush(stdout);

tcgetattr(0, &Otty);

Ntty = Otty;

Ntty.c\_iflag = 0;

Ntty.c\_oflag = 0;

Ntty.c\_lflag &= ~ICANON;

#if 1

Ntty.c\_lflag &= ~ECHO;

#else

Ntty.c\_lflag |= ECHO;

#endif

Ntty.c\_cc[VMIN] = 0;

Ntty.c\_cc[VTIME] = 1;

#if 1

#define FLAG TCSAFLUSH

#else

#define FLAG TCSANOW

#endif

if (0 == (error = tcsetattr(0, FLAG, &Ntty)))

{

error = read(0, &ch, 1);

error += tcsetattr(0, FLAG, &Otty);

}

return (error == 1 ? (int)ch : -1);

}

**(5). int Getcommand() : getch() 함수를 호출해 얻은 입력 키 값에 따라 적절한 값을 반환해준다. 현재 사용자의 입력키가 ‘d’ / ‘D’ 인 경우 0을, ‘w’ / ‘W’ 인 경우 1을, ‘a’ / ‘A’ 인 경우 2를 그리고 ‘s’ / ‘S’ 인 경우 3을 반환해주도록 설정되어 있다.**

(기존에 제공되었던 코드에서 수정하지 않음)

int GetCommand() {

int ch = getch();

switch (ch)

{

case 'd':

case 'D': return 0;

case 'w':

case 'W': return 1;

case 'a':

case 'A': return 2;

case 's':

case 'S': return 3;

default: return -1;

}

return -1;

}

**(6). int move\_up (int b[4][4]) : 기존 코드에는 없던 user\_func\_1이다. 블록을 위쪽으로 몰아주는 함수이다. 이 함수는 마지막에 합쳐지거나 이동한 횟수를 나타내는 flag인 count를 반환한다. 만약 count가 0이면 합쳐지거나 이동하지 않았다는 의미이다. 이 함수는 반복문을 이용하여 배열을 위에서부터 아래로 다음 과정을 체크한다.**

**1. 우선 만약 현재 배열 값이 0이 아니면 같은 col의 아래 숫자들과 비교한다. 처음으로 나온 0이 아닌 숫자가 현재 배열 값과 같은 값이면 위로 보내서 합쳐준다. 합쳐진 아래 숫자의 자리에는 0을 넣는다. 만약 처음으로 나온 0이 아닌 숫자가 같은 값이 아니면 break 한다.**

**2. 다음으로는 현재 배열 값과 같은 col의 위의 숫자들을 훑어서 처음으로 0이 아닌 숫자가 나올 때 break한다. 그 지점에서 1을 다시 더한 값이 현재 값과 같이 않다면 위로 올라갈 자리가 있다는 뜻이다. 따라서 위로 보내주고 마찬가지로 원래 자리에는 0을 채워준다.**

**이렇게 하면 위에서부터 아래로 내려가며 확인하기 때문에 위에서 한번 합쳐진 숫자는 밑의 수와 같더라도 다시 합쳐지지 않는다.**

int move\_up (int b[4][4])

{

int i,j,temp;

int count=0;

for(i=0; i<4; i++)

{

for(j=0; j<4; j++)

{

if(b[i][j] != 0)//b[i][j]가 0이 아닐 때 아래 숫자와 비교해서 같으면 합쳐주고 아래 배열은 0으로 만들어준다.

{

for(temp=i+1; temp<4; temp++)

{

if(b[temp][j]==b[i][j])

{

b[i][j] \*= 2;

b[temp][j]=0;

count++;

break;

}

else if(b[temp][j]!=0)

break;

}

for(temp=i-1; temp>=0; temp--)//위의 숫자들을 훑으면서 0이 아닌 숫자가 나오면 break한다.

{

if(b[temp][j]!=0)

break;

}

temp++;

//break한 지점에서 1을 더해준 index가 i와 같지 않으면 위로 올라갈 자리가 있다는 뜻이므로 올려준다.

if(i!=temp)

{

b[temp][j]=b[i][j];

b[i][j]=0;

count++;

}

}

}

}

return count; //count는 움직이거나 합쳐진 횟수를 세어준다. 만약 count가 0일 경우 움직이지 않고 합쳐지지 않았다는 뜻이므로 main에서 0을 반환하여 랜덤 숫자를 생성하지 않는다.(과정2의 추가기능 구현)

}

**(7). int move\_down(int b[4][4]) : 기존 코드에는 없던 user\_func\_2이다. 블록을 아래쪽으로 몰아주는 함수이다. 방향성만 반대이고 알고리즘은 move\_up 함수와 같다.**

int move\_down(int b[4][4])

{

int i,j,temp;

int count=0;

for(i=3; i>=0; i--)

{

for(j=3; j>=0; j--)

{

if(b[i][j] != 0)//b[i][j]와 위의 숫자를 비교해서 같은것이 있으면 합쳐준 후 위 배열은 0으로 만들어준다.

{

for(temp=i-1; temp>=0; temp--)

{

if(b[temp][j]==b[i][j])

{

b[i][j] \*= 2;

b[temp][j]=0;

count++;

break;

}

else if(b[temp][j]!=0)

break;

}

for(temp=i+1; temp<4; temp++)//아래의 숫자들을 훑으면서 0이 아닌 숫자가 나올때 break해준다.

{

if(b[temp][j]!=0)

break;

}

temp--;

//break한 지점에서 1을 빼준 index가 i와 같지 않으면 아래로 내려갈 자리가 있다는 뜻이므로 내려준다.

if(i!=temp)

{

b[temp][j]=b[i][j];

b[i][j]=0;

count++;

}

}

}

}

return count;//move\_up의 count기능과 같다.

}

**(8). void rotate(int b[4][4]) : 판을 시계방향으로 90도 회전시켜주는 함수이다. 새로운 배열 c[4][4]를 선언해서 b의 내용을 복사해둔 후 c를 회전시켰을 때 값이 들어가는 모양을 b에 저장해준다.**

void rotate(int b[4][4]){

/\* user code \*/

int i,j;

int c[4][4];

for(i=0; i<4; i++)

{

for(j=0; j<4; j++)

c[i][j]=b[i][j];

}//c배열을 하나 선언해서 b를 복사한다.

init();//b[][]배열을 비운다.

for(i=0; i<4; i++)

{

for(j=0; j<4; j++)

b[j][3-i]=c[i][j];//c배열(==b배열)을 90도 돌렸을때를 b에 다시 저장한다.

}

}

**(9). int set\_board(int dir, int b[4][4]) : 입력한 키(dir)와 배열을 받아와서 키에 맞게 이동시켜주는 함수이다. 아이디어는 다음과 같다. 배열을 위, 아래로 몰아주는 함수만 구현하면 좌우로 이동시킬 때는 그 함수들과 rotate함수를 적절히 배치시켜 간단히 구현할 수 있다. 또한 move\_up 과 move\_down 함수에서 배열이 움직이거나 숫자가 합쳐진 횟수를 반환시키므로 이 값을 받아와서 flag처럼 사용할 수 있다. 이 함수는 정상적인 키가 받아지고 (flag로 판단) 배열이 움직이거나 숫자가 합쳐졌으면 (count로 판단) 1을 반환하고, 그 경우가 아니라면 0을 반환한다. main에서 이 함수가 1을 반환할 때만 랜덤으로 수를 발생시켜 새로운 필드를 그리게끔 설정하였다.**

int set\_board(int dir, int b[4][4]){

/\* user code init \*/

//idea : 배열을 위,아래로 몰아주는 함수(move\_up)(move\_down) 를 구현하면 양 옆으로 이동시킬 때는 rotate함수를 이용하여 간단히 구현할 수 있다.//

int flag=1;

int count;

if(dir==0)//right키

{

rotate(b);

count=move\_down(b);

rotate(b);

rotate(b);

rotate(b);

}

else if(dir==1)//up키

{

count=move\_up(b);

}

else if(dir==2)//left키

{

rotate(b);

count=move\_up(b);

rotate(b);

rotate(b);

rotate(b);

}

else if(dir==3)//down키

{

count=move\_down(b);

}

else//정상 키가 입력되지 않았을 때

flag=0;

if(flag==1 && count!=0)

return 1; //정상적인 방향키 입력, 움직이거나 합쳐졌음.

else if(flag ==0 || count==0)

return 0; //정상적인 방향키 입력이 되지 않음, 움직이지도 합쳐지지도 않음.

}

**(10). int is\_game\_over() : 게임이 오버했는지 아닌지를 판단해주는 함수이다. 우선 반복문을 돌려서 배열을 모두 확인하여 0이 아닌 숫자가 있을 때 마다 count++을 해준다. 만약 최종 count가 16이면 배열이 모두 차있다는 뜻이다. 다음으로 각 배열의 인접한 곳과 비교하였을 때 같은 값이 있다면 same\_flag를 1로 설정해준다. same\_flag가 0이면 인접한 곳에 같은 값이 없다는 뜻이다. 따라서 count가 16이면서 same\_flag가 0이면 게임오버이므로 1을 반환하고 그렇지 않으면 0을 반환한다. main에서 전역변수 game\_over가 이 함수의 반환 값을 받으므로 game\_over가 1이 될 때 게임을 종료한다.**

int is\_game\_over(){

/\* user code \*/

/\* if game over return 0, else then return 1 \*/

int i,j,count=0;

int ti,tj;

int ci,cj;

int same\_flag=0;

srand(time(NULL));

for(i=0; i<16; i++)

{

if(b[i/4][i%4] !=0)

count++;

}//반복문을 돌리면서 배열이 0이 아닐 때마다 count++, count가 16이면 꽉 차있는 것.

for(i=0; i<4; i++)

{

for(j=0; j<4; j++)

{

ti=i-1;

tj=j-1;

ci=i+1;

cj=j+1;

if(ti>0 && b[i][j]==b[ti][j])

same\_flag=1;

if(tj>0 && b[i][j]==b[i][tj])

same\_flag=1;

if(ci<4 && b[i][j]==b[ci][j])

same\_flag=1;

if(cj<4 && b[i][j]==b[i][cj])

same\_flag=1;

}

}//반복문을 돌리면서 인접한 위치에 같은 숫자가 있으면 same\_flag를 1로 설정

if(count==16 && same\_flag==0)

return 1;//게임오버이다; 판이 꽉 차있으며 same\_flag가 0이다(움직일 수 없다)

else

return 0;//게임오버아니다; 판이 꽉 차있지 않거나 same\_flag가 1이다(움직일 수 있다)

}

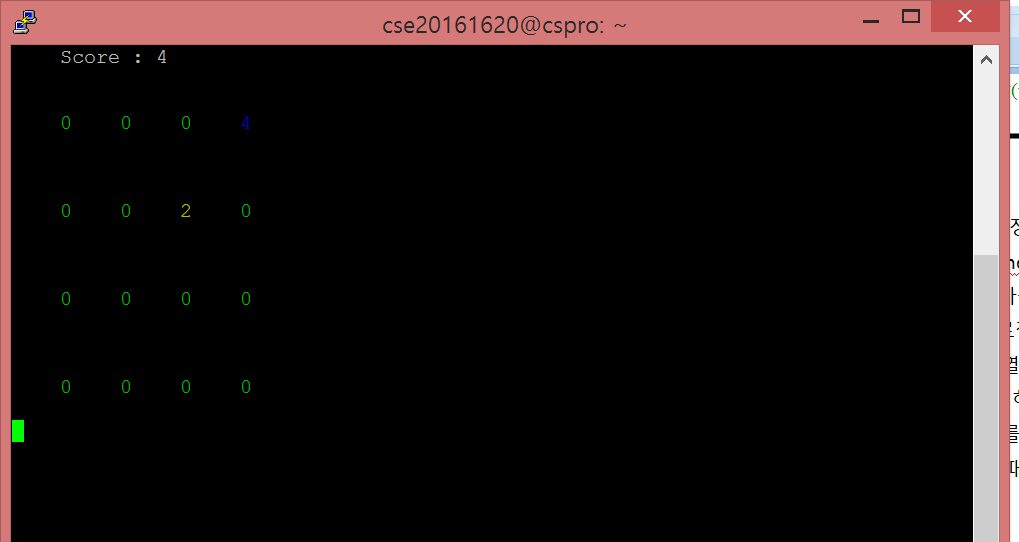
**2-3. 2(또는 4)의 확률을 설정한 방법 (보고서 요구사항)**

rand() 함수는 0부터 stdlib.h 파일에 RAND\_MAX로 정의된 값까지의 정수 범위에서 난수 (랜덤한 값)를 발생시킨다. 호출할 때마다 다른 난수를 생성하기 위해서 srand() 함수의 인자로 time() 함수가 반환하는 값을 지정해준다. rand함수의 응용으로 다음과 같이 가능하다. rand()%N 이라고 호출할 경우 0부터 N-1 까지의 숫자 중에서 난수를 발생시킨다. 이 프로젝트에서는 2를 2/3의 확률로, 4를 1/3의 확률로 발생시켜야 하므로 배열 arr[3]을 선언하고, 배열의 원소로 {2,2,4}를 지정해 주었다. 그 다음 랜덤으로 발생시킨 수를 저장하는 변수를 num이라 하고 **num=arr[rand()%3]**과 같이 호출해 주었다. 즉 rand()%3에 의해 0,1,2 중에서 랜덤으로 수를 생성하며, 이 값이 배열의 인덱스로 들어가므로 배열의 원소인 2와 4중에서 수를 불러온다. 이때 원소의 구성은 2가 2개, 4가 1개이므로 2는 2/3의 확률로, 4는 1/3의 확률로 발생시킬 수 있다.

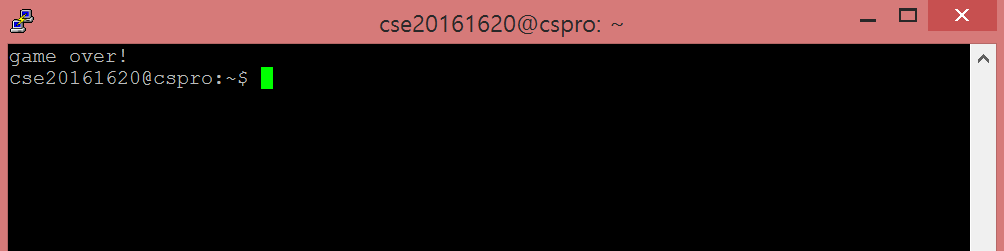
**기초공학설계(3반) 20161620 이수연**

**3. 실행화면**

1) 초기 화면 : 난수 2개가 랜덤한 위치에 설정

****

2) 게임오버 화면 : 콘솔 창을 지우고 game over! 출력

****

3) 게임오버가 된 화면 (스크롤 위로 올림) : 판이 모두 차있고 인접한 숫자가 서로 같지 않음

****

**기초공학설계(3반) 20161620 이수연**

**4. 프로그램의 완성도**

**\* 2(또는 4)가 정확히 설정이 되었는가?**

-> 배열 arr[3]을 선언하고, 배열의 원소로 {2,2,4}를 지정해 주었다. 랜덤으로 발생시킨 수를 저장하는 변수를 num이라 하고 num=arr[rand()%3]과 같이 호출해 주면 rand()%3에 의해 0,1,2 중에서 랜덤으로 수를 생성하며, 이 값이 배열의 인덱스로 들어가므로 배열의 원소인 2와 4중에서 수를 불러온다. 이때 원소의 구성은 2가 2개, 4가 1개이므로 2는 2/3의 확률로, 4는 1/3의 확률로 발생시킬 수 있다.

**\* 올바르게 이동하는가?**

-> rotate를 어떻게 활용할 지 고민하다가 위로 몰아주는 move\_up 함수와 아래로 몰아주는 move\_down 함수를 추가로 구현해주었다. 두 함수는 위(아래)에서부터 아래(위)로 내려가며(올라가며) 확인하기 때문에 한번 합쳐진 숫자는 다음의 수와 같더라도 다시 합쳐지지 않는다. Up의 경우에는 move\_up 함수만 호출해주고 Down의 경우에는 move\_down 함수를 호출한다. Right의 경우에는 시계방향으로 90도 회전한 후(rotate) 아래로 몰아준다. 그 다음 rotate를 세 번 호출하면 오른쪽으로 몰아준 효과를 낼 수 있다. Left의 경우에는 시계방향으로 90도 회전한 후(rotate) 위로 몰아준다. 그 다음 rotate를 세 번 호출하면 왼쪽으로 몰아준 효과를 낼 수 있다. 또한 move\_up과 move\_down 함수는 이동하지 않고 합쳐지지도 않으면 flag가 0이 되고 이를 반환한다. 이 조건을 set\_board에 추가하여 키를 눌러도 이동할 곳이 없고 합쳐지지도 않을 때는 난수가 생성되지 않도록 하였다. (과정2 추가구현) 마지막으로 정상 키가 아닐 때는 이동하지 않는다. 따라서 이에 따라 움직인다면 올바르게 이동할 수 있다.

**\* 게임오버는 정확한 때에 되는가?**

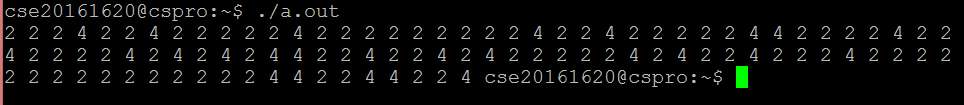
-> 반복문을 돌리면서 판의 숫자를 모두 확인하여 0이 아닌 숫자가 있을 때 마다 count++을 해준다. 만약 최종 count가 16이면 배열이 모두 차있다는 뜻이다. 다음으로 각 배열의 인접한 곳과 비교하였을 때 같은 값이 있다면 same\_flag를 1로 설정해준다. 이때 임시 위치들을 설정하여 판을 벗어났을 때는 검사하지 않도록 설정해준다. same\_flag가 0이면 인접한 곳에 같은 값이 없다는 뜻이다. 따라서 count가 16이면서 same\_flag가 0이면 게임오버이다. 이렇게 flag를 두 개 사용하면 만약 필드가 다 차있더라도 인접한 곳에 같은 수가 있으면 게임 오버되지 않고 계속 합쳐줄 수 있다. 이와 같이 설정하여 game\_over 변수에 값을 반환해주기 때문에 올바르게 게임오버가 된다.

**기초공학설계(3반) 20161620 이수연**

**5. 시험 및 평가 방식 분석**

다음과 같은 차례로 만든 프로그램을 시험해보았다. 우선 프로그램의 기본적인 사항이 지켜졌는지 확인하였다. 초기화면에서 랜덤한 위치에 난수가 2개 생성되는지, 입력한 방향키대로 판이 움직이는지, 판의 가장 높은 점수가 실시간으로 업데이트 되는지를 체크하였다. 그 다음 종료 조건이 만족되었는지 확인했다. 처음에는 인접한 곳에 같은 숫자가 있어도 판이 꽉 차면 무조건 종료하였다가 output.txt 에 입력되는 과정을 검토하며 오류가 있음을 깨달았다. 그래서 종료 조건에 ‘인접한 곳에 같은 숫자가 없을 때까지’를 추가하였다. 그 다음 ppt에 제시되어있던 과정 2의 추가 구현을 하였다. 처음에는 이동했는지 여부와 합쳐졌는지 여부를 판단하는 flag가 없었고, 그 때문에 움직이거나 합쳐지지 않아도 난수가 발생하였었다. 이를 보완하기 위해 flag를 설정해주었다. main함수에서 종료되었는지 확인하는 is\_game\_over함수를 처음에는 Getcommand의 루프 마지막에 넣어주었었다. 그러나 그렇게 하면 종료되어야 할 때 어떠한 키를 한번 더 눌러야만 종료된다는 것을 깨닫게 되었고, 루프 밖으로 함수를 빼서 수정하였다. draw\_board함수에서 전체 생성된 숫자의 개수가 변수 tot과 같은지 확인하는 부분이 있었는데 처음에는 그 부분이 없어도 게임이 잘 되었기 때문에 해주지 않았지만 프로젝트의 요구사항이었기 때문에 추가하였다. 마지막으로 실행만 해보아서는 난수가 제대로 된 확률로 나타나는지 알 수가 없어서 그 부분의 함수만 따로 돌려보면서 난수를 여러 번 프린트하며 시험해 보았다. 이러한 모든 test case를 거치면서 프로젝트를 계속 수정하고 보완하였기 때문에 프로젝트의 완성도가 높을 것이라고 예상된다.

<실제로 난수를 시험해본 화면 - 100개의 난수 출력>



**기초공학설계(3반) 20161620 이수연**

**6. 참고사항**

서강대학교 컴퓨터공학과 기초공학설계 프로젝트 강의 교재 (ppt)

**7. 팀 구성**

이수연 100%

**8. 수행기간**

2017년 5월 15일 (월요일) - 2017년 6월 13일 (화요일)

**기초공학설계(3반) 20161620 이수연**