CSE3013 컴퓨터공학 실험: 테트리스 프로젝트

담당교수: 서강대학교 컴퓨터공학과 장형수

분반 : 3반 학번 : 20151616 이름 : 최승환

1. 설계 문제 및 목표

ncurses 라이브러리가 제공하는 리눅스 터미널 상에서의 GUI를 이용하여 테트리스 게임 프로그램을 제작하고 여기에 랭킹 시스템과 블록 배치 추천 기능을 추가 구현한다.

테트리스 게임은, 블록을 90도씩 반시계 방향으로 회전하거나 세 방향(좌, 우, 하)으로 움직여 필드에 쌓으면, 빈틈없이 채워진 줄은 지워지고 그에 따른 스코어를 얻어 결과적으로 가장 높은 최종 스코어를 얻는 것이 목적이다.

프로그램을 실행하면 사용자는 메뉴를 선택할 수 있다. 1번을 선택하면 테트리스 게임을 플레이할 수 있고, 2번을 선택하면 랭킹 정보를 확인할 수 있고, 3번을 선택하면 블록 배치 추천 기능을 따라 플레이되는 모드로 테트리스 게임이 실행되고, 마지막으로 4번을 선택하면 프로그램이 종료된다.

(이 내용까지는 그대로 두고 1주차, 2주차, 3주차에는 어떤 것을 해야 되는 지 기술 할 것.)

1주차는 테트리스의 play부분 즉, 실제 테트리스를 유저가 플레이할 수 있는 부분을 구현한다. 이 경우 각 블럭이 좌우로 움직이고 회전하는 것, 아래로 내려가는 것과 1초에 한 번씩 블럭이 내려가는 기능을 구현해야 한다. 또한 1초에 한 칸씩 블럭이 내려가는 기능을 통해 점수를 계산하고, 모두 채워진 줄이 있는 경우 해당 줄을 지우는 작업까지가 1주차의 내용에 포함된다. 이 때 점수를 합산하는 방식에 블럭이 땅에 닿는 면적과 지워진 줄의 개수를 고려한다.

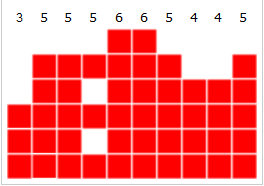
2주차는 테트리스를 플레이하고 난 후 저장한 점수를 랭킹으로 표현하는 부분이다. 이 부분에서는 링크드리스트를 사용해 플레이가 종료될 때마다 그 점수를 저장하고, 이를 대소를 비교해 랭킹에 삽입한다. 이 랭킹을 원하는 구간만큼 보도록 만들고, 이 중에 원하는 랭킹에 해당하는 플레이어만 삭제하는 기능도 여기에 포함된다.

3주차는 테트리스의 추천기능을 구현해야 한다. 알고리즘을 통해 어떤 곳에 두는 것이 최적의 플레이인지를 판단하고 이를 플레이어에게 표시해주어야 한다. 이에는 트리를 이용해 모든 경우의 수를 다 고려하는 방법이 있다. 이 때는 2개 혹은 3개의 수를 미리 예측해 가장 많은 점수를 낼 수 있는 위치에 블럭을 표시하도록 한다. 마지막으로, 테트리스의 추천기능을 확장한 테트리스 자동플레이 기능을 구현해야 한다. 자동플레이 기능에는 현 상황에서 최적의 플레이를 계산하고, 이 플레이를 자동으로 실행하도록 하고, 이를 필드에 표시한다.(이 경우에는 트리가 아닌 가중치 방식을 채용했다.)

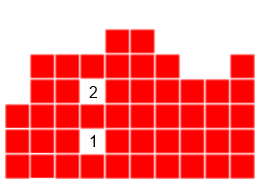
2. 요구사항

2.1 설계 목표 설정

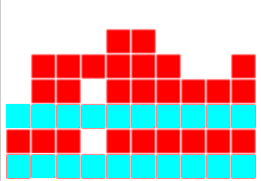
테트리스 자동플레이 기능의 구현 설계목표



그림과 같이 필드가 구성되어 있다고 하자. 이 경우 높이를 계산해준다. 이 높이에 특정 상수를 부여해 너무 높은 필드를 구성하지 않도록 계산해준다. 높이를 계산하면서 하나 더 고려해줘야 할 것이 있는데 이것이 고름의 정도이다. 블럭이 너무 들쑥날쑥한 경우도 가중치를 줘서 피하도록 만들어줘야 한다.



또, 필드의 상태를 해석할 때 하나 고려해야 할 것이 구멍의 숫자이다. 필드 내에 있는 구멍의 숫자를 셀 때는 위와 같이 윗면에 블럭이 존재하면서 해당 칸에는 블럭이 없는 좌표를 세어 줘야 한다. 이 또한 가중치를 주도록 하자.



마지막으로 블럭을 놓았을 때 지워지는 줄 수를 세어줘야 한다. 줄이 지워지는 경우를 계산해 이에 대한 가중치를 주어준다. 줄이 지워지는 경우는 다른 경우들과는 다르게 지향해야 할 사항이다. 따라서, 이 경우에 대해서는 양의 가중치를 주도록 한다.

2.2 합성

테트리스 게임 프로그램의 설계 및 구현을 위해 요구되는 이론, 자료구조, 알고리즘 등을 조사 분석하여 전체적인 설계를 수행한다.(2.1의 요구사항을 해결하기 위한 효율적인 알고리즘 및 자료구조를 구상하여 기술할 것.)

먼저 테트리스 필드를 저장할 방법부터 시작해보면, 테트리스의 현재 필드를 저장할 방법이 필요하다. 이를 위해, 필드의 크기만큼의 배열을 선언한다. 이 배열 안에는 해당 좌표가 블럭이 있는지 없는지를 확인할 수 있어야 한다. 이를 위해 좌표에 블럭이 있는 곳이라면 1을 저장해 주고 블럭이 없는 곳이라면 0을 저장해준다. 이를 이용하면 현 블럭을 필드에 옮길 수 있는지를 확인해줄 수 있다. 좌표를 확인해 블럭의 좌표가 1인 곳에 필드의 좌표도 1이면 해당 좌표로 옮기는 것이 불가능하다.

랭크를 구현하기 위해서 사용한 자료구조는 linked list이다. linked list를 이용하면 삽입과 삭제가 간편하기 때문에 간편하게 랭크를 짤 수 있다. 또한 linked list를 이용해 삽입과 삭제를 하는 동시에 정렬이 가능하기 때문에 별다른 정렬 알고리즘 없이 빠르게 삽입할 수 있다.

마지막으로 설명할 부분은 자동 플레이 알고리즘이다. 자동 플레이 알고리즘으로 점수를 메기기에 앞서 필드의 정보를 가져와야 한다. 필드의 정보는 현 필드의 정보와 블럭을 놓을 위치에 따라 변한 정보를 모두 포함한다. 이 경우 x 좌표를 0에서 9까지 늘리면서 각 x좌표마다 내려서 도착할 수 있는 위치에 블럭을 저장하고 이 필드를 저장해 자동플레이 알고리즘의 가중치를 이용한다. 자동 플레이 알고리즘에서는 앞서 말했던 우선순위들을 이용해 한 필드에 대한 점수를 메긴다. 그리고 이들 중 가장 높은 가중치를 가진 필드를 현 필드에 옮겨 붙인다.

2.3 분석

newRank는 새로운 노드를 삽입할 때 사용하는 함수이다. 노드를 삽입하기 위해서 head부터 마지막 노드까지 하나씩 확인하며 삽입을 하게 되는데, 이 때 효율성을 위해서 삽입을 할 때 크기를 고려해 삽입하게 된다. 또한 이 코드의 경우 삽입과 정렬이 끝난 후 다시 노드들을 보면서 프린트를 해주는 대신에 삽입을 하는 동시에 위치를 찾으면 출력을 하고 삽입과 동시에 처리한다.

void newRank(int score){

Rank\_node\* currnode;

Rank\_node\* bef=NULL;

Rank\_node\* newnode = malloc(sizeof(Rank\_node));

int ncnt,flag=1;

FILE\* wfp;

//FILE\* rfp = fopen("rank.txt","r");

clear();

printw("enter your name : ");

scanw("%s",newnode -> name);

newnode -> score = score;

newnode -> next\_node = NULL;

createRankList();

wfp = fopen("rank.txt","w");

currnode = head;

ranknum++;

fprintf(wfp,"%d\n",ranknum);

if(ranknum == 1)

{

fprintf(wfp,"%s %d\n",newnode->name,newnode->score);

head = newnode;

return;

}

for(ncnt=0;ncnt<ranknum-1;++ncnt)

{

if(currnode -> score < newnode -> score)

{

if(currnode != head)

bef -> next\_node = newnode;

newnode -> next\_node = currnode;

currnode = newnode;

//fprintf(wfp,"%s %d\n",newnode->name,newnode->score);

break;

}

fprintf(wfp,"%s %d \n",currnode -> name,currnode -> score);

bef = currnode;

currnode = currnode -> next\_node;

}

if(currnode == NULL)

{

return;

}

while(currnode != NULL)

{

fprintf(wfp,"%s %d\n",currnode -> name,currnode -> score);

if(currnode -> next\_node == NULL)

break;

currnode = currnode -> next\_node;

}

currnode = currnode -> next\_node;

fclose(wfp);

return;

// user code

}

또한 가중치의 경우 앞서 설명한 구멍의 개수, 사라질 줄의 개수, 높이, 고른 정도에 따라 주는 상수가 있다. 이 상수의 경우, 테트리스 알고리즘에 최적화된 상수를 가져와 사용한다. a,b,c,d에 이 상수들을 저장해두고, 각각 이를 구하는데 사용된 네 가지 함수

checkheight

checkholes

checkcompletelines

를 사용해 구한다.

특히 height의 경우 각 블럭들의 높이를 구하기 때문에 고른 정도를 뜻하는 bumpiness를 여기에서 같이 계산해 저장해준다.

int checkheight(char newfield[HEIGHT][WIDTH],int\* bumpiness)

{

int ccnt,rcnt;

int height=0,befheight=0,maxh=0;

\*bumpiness = 0;

for(ccnt=0;ccnt<WIDTH;++ccnt)

{

befheight = height;

height=0;

for(rcnt=0;rcnt<HEIGHT;++rcnt)

{

if(newfield[rcnt][ccnt] == 1)

{

break;

}

}

height = HEIGHT-rcnt;

if(height-befheight > 0)

{

\*bumpiness += (height-befheight);

}

else

{

\*bumpiness -= (height-befheight);

}

if(maxh<height)

{

maxh = height;

}

}

return maxh;

}

int checkholes(char newfield[HEIGHT][WIDTH])

{

int ccnt,rcnt,flag=0;

int holes=0;

for(ccnt=0;ccnt<WIDTH;++ccnt)

{

flag=0;

for(rcnt=0;rcnt<HEIGHT;++rcnt)

{

if(newfield[rcnt][ccnt])

{

flag=1;

}

else if(flag == 1)

{

//flag=0;

holes++;

}

}

}

return holes;

}

int checkcompletelines(char newfield[HEIGHT][WIDTH])

{

int delete\_num=0;

int ycnt,xcnt,clear\_flag;

for(ycnt=0;ycnt<HEIGHT;++ycnt)

{

clear\_flag=1;

for(xcnt=0;xcnt<WIDTH;++xcnt)

{

if(newfield[ycnt][xcnt] == 0)

{

clear\_flag=0;

break;

}

}

if(clear\_flag == 1)

{

++delete\_num;

}

}

return delete\_num;

}

2.4 제작

해당 좌표로 블럭이 이동할 수 있는지를 판단하는 CheckToMove 함수

int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){

// user code

int xcnt,ycnt;

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

{

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

{

if(block[currentBlock][blockRotate][ycnt][xcnt] == 1)

{

if(f[blockY+ycnt][blockX+xcnt] == 1)

{

return 0;

}

if(blockY+ycnt>HEIGHT-1)

{

return 0;

}

if(blockX+xcnt>WIDTH-1)

{

return 0;

}

if(blockX+xcnt<0)

{

return 0;

}

}

}

}

return 1;

}

랭크 시스템에서 랭크를 프린트 하는 print\_node 함수와 랭크에서 지우는 delete\_node 함수

void print\_node()

{

FILE\* fp = fopen("rank.txt","w");

Rank\_node\* currnode = head;

fprintf(fp,"%d\n",ranknum);

if(currnode == NULL)

{

return;

}

while(currnode -> next\_node != NULL)

{

fprintf(fp,"%s %d\n",currnode -> name,currnode -> score);

currnode = currnode -> next\_node;

}

fprintf(fp,"%s %d\n",currnode -> name,currnode -> score);

fclose(fp);

return;

}

void delete\_node(int temp)

{

//todo

Rank\_node\* currnode = head;

Rank\_node\* befnode;

int cnt=temp;

int chk = 0;

echo();

ranknum--;

wgetch(stdscr);

while(temp > 1)

{

temp--;

befnode=currnode;

currnode = currnode -> next\_node;

}

if(currnode -> next\_node == NULL)

{

if(cnt == 1)

{

free(currnode);

head = NULL;

}

else

befnode -> next\_node =NULL;

}

else if(cnt == 1)

{

head = currnode -> next\_node;

free(currnode);

}

else

{

befnode -> next\_node = currnode -> next\_node;

//free(currnode);

}

noecho();

}

자동플레이를 하도록 가중치와 점수를 주는 recommender 함수

void recommender(int sig/\*RecNode \*root\*/){

int maxy=0,maxx=0,maxc=0,maxb=0;

int rcnt,ccnt=0,bcnt;

int flag=0;

int height=0,bumpiness=0,holes=0,completelines=0;

int r2=0,c2=0;

char newfield[HEIGHT][WIDTH];

double maxscore=0;

double a=-0.510066;

double b = 1.760666;

double c = -0.55663;

double d = -0.184483;

if(!CheckToMove(field,nextBlock[0],blockRotate,blockY+1,blockX))gameOver=1;

else

{

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

{

for(ccnt=0;ccnt<WIDTH;++ccnt)

{

height=0;

holes=0;

completelines=0;

for(r2=0;r2<HEIGHT;++r2)

{

for(c2=0;c2<WIDTH;++c2)

{

newfield[r2][c2] = field[r2][c2];

}

}

//cpfield(newfield);

rcnt=0;

while(CheckToMove(newfield,nextBlock[0],bcnt,++rcnt,ccnt));

--rcnt;

AddBlockToField(newfield,nextBlock[0],bcnt,rcnt,ccnt);

height = checkheight(newfield,&bumpiness);

holes = checkholes(newfield);

completelines = checkcompletelines(newfield);

if(flag == 0)

{

flag = 1;

maxscore = a\*height+b\*completelines+c\*holes+d\*bumpiness;

}

if(maxscore<a\*height+b\*completelines+c\*holes+d\*bumpiness)

{

maxscore = a\*height+b\*completelines+c\*holes+d\*bumpiness;

maxx = ccnt;

maxy = rcnt;

maxb = bcnt;

}

}

}

for(r2=0;r2<HEIGHT;++r2)

{

for(c2=0;c2<WIDTH;++c2)

{

newfield[r2][c2] = field[r2][c2];

}

}

}

if(maxx == 0)

{

rcnt=0;

while(CheckToMove(field,nextBlock[0],maxb,++rcnt,maxx));

--rcnt;

score+=AddBlockToField(field,nextBlock[0],maxb,rcnt,maxx);

}

else

{

score+=AddBlockToField(field,nextBlock[0],maxb,maxy,maxx);

}

score+=DeleteLine(field);

PrintScore(score);

DrawField();

nextBlock[0] = nextBlock[1];

nextBlock[1] = nextBlock[2];

nextBlock[2] = rand()%7;

DrawNextBlock(nextBlock);

DrawBlockWithFeatures(blockY,blockX,nextBlock[0],blockRotate);

timed\_out = 0;

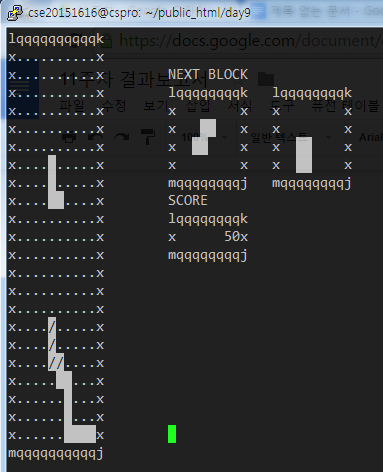
return;

}

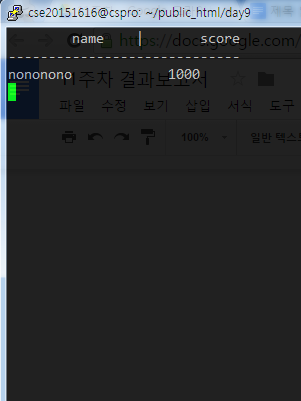
2.5 시험

위의 과정에서 수행한 문제 정의, 프로그램 순서도와 순서도 상의 각 부분 역할 및 구현 방법, 프로그램의 구현 방법, 구현 방법의 이론과의 연관성, 구현한 프로그램의 내용, 수행화면 등을 정리한다.

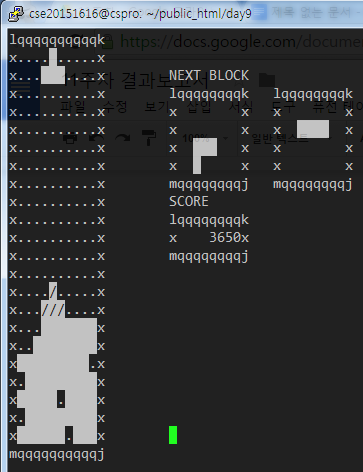
play 함수를 이용해 테트리스를 플레이하는 사진



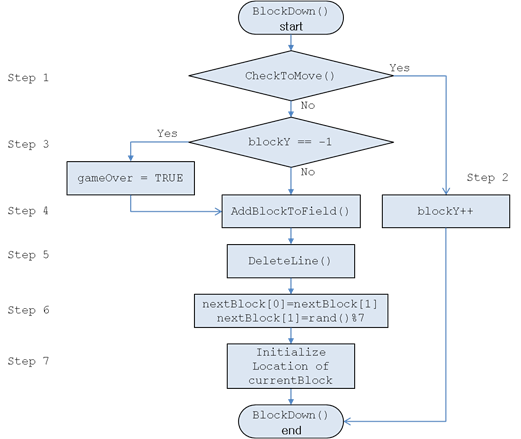
랭크를 표시하는 경우



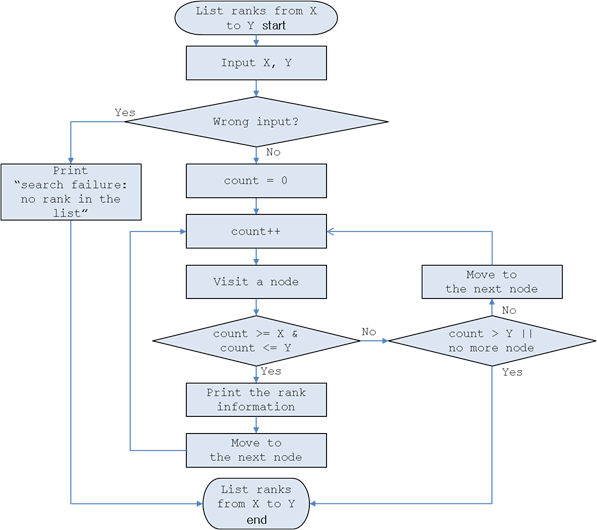
recommendedPlay 함수를 통해 자동플레이를 하는 경우



2.6 평가



BlockDown 함수는 act.sa\_hander를 통해 1초에 한 번씩 호출된다. 이 때 CheckToMove를 이용해 블럭이 한 칸 아래로 내려갈 수 있는지를 확인한다. 내려갈 수 있는 경우 한 칸 아래로 이동시키고, 그렇지 않은 경우 해당 블럭을 AddBlockToField 함수를 이용해 필드에 추가해준다. 그 후 다음 블럭을 현재 블럭으로 옮겨주고 점수를 계산해준다.



랭크를 메기는 함수로는 삽입하는데 쓰이는 newRank 함수, 노드들을 print하는데 쓰이는 print\_node 함수, 노드를 삭제하는데 사용되는 delete\_node함수가 있다. 이는 랭크를 보는 화면에서 입력받은 숫자에 따라 맞는 함수를 호출하면 된다.

2.7 환경

학생들이 리눅스 서버를 접속하여 프로젝트를 진행하므로 해당 서버에 접속할 수 있는 데스크탑과 ssh 접속 프로그램을 제공한다. 접속하는 리눅스 서버에 각 학생들에게 하위 계정을 발급하여 할당받는 용량에 한하여 자유롭게 이를 이용하여 프로젝트를 진행할 수 있는 환경을 제공한다.

2.8 미학

AddBlockToField 함수의 경우 field를 하나의 인자로 받는다. 그리고 해당 인자에 해당하는 필드에 블럭을 추가해주는데, 이 이유로 이 함수는 전역변수인 field에 block을 추가하는(BlockDown 함수의 경우) 데에도 사용할 수 있고, newfield에 block을 추가할 때(recommender 함수의 경우)에도 사용할 수가 있다. 또한 CheckToMove의 경우에도 필드를 인자로 받아 blockdown이나 CheckToMove 함수와 같은 함수에서 같이 사용할 수 있다.

2.9 보건 및 안정

자동플레이 함수의 경우 잘못된 위치에 놓이거나 공중에 블럭이 위치하는 경우가 있었다. 이 경우 gdb를 이용해 대부분 디버깅했고, 공중에 블럭이 위치하는 경우는 블럭이 닿는 면적 없이 공중에 떠있는 경우 다시 아래로 쭉 내리는 예외처리를 이용해 오류를 잡았다.

3. 기 타

3.1 환경 구성

해당 테트리스의 코드 길이는 약 1000줄로 지금까지 진행했던 프로젝트들에 비해서 꽤 긴 편이다. 이런 코드를 단순 printf나 printw를 이용해 출력하는 방식으로 디버깅을 한다면 굉장히 긴 시간이 걸릴 것이다. 이를 해결하기 위해 gdb 디버깅을 사용했다. gdb에서는 한 줄에 도달했을 때 breakpoint를 걸어주는 것과 함수에 도달했을 때 breakpoint를 거는 것, 심지어 어떤 변수의 상태에 따라서 breakpoint를 거는 것도 가능하다. 하지만 이를 사용하기 위해서는 컴파일에 옵션을 걸어줘야 하는데 이것이 -g 옵션이다. 따라서 이를 이용하기 위해서 makefile 내에서 gcc -g tetris.c 와 같은 방식으로 컴파일하도록 하자.

3.2 참고 사항

자동플레이의 경우에는 기존의 트리 방식이 아닌 가중치 방식을 사용했다.

3.3 팀 구성

최승환 100%

3.4 수행기간

5.4~5.24