|  |
| --- |
|  |
| **컴퓨터공학실험1 (1반) 9주차 결과보고서 (테트리스(1))** |
|  |
|  |
|  |
| 컴퓨터공학과 20161620 이수연 |

**컴퓨터공학실험1 // 테트리스(1) 결과보고서**

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

**1. 실습시간에 작성한 프로그램의 함수들이 예비보고서에서 작성한 각 구현 함수들의 pseudo code와 어떻게 달라졌는지 설명하고, 각 함수에 대한 시간 및 공간 복잡도를 보이시오 (각 함수의 시간 및 공간복잡도를 구할 때, 어떤 변수에 의존하는지를 판단해야 한다.)**

|  |  |
| --- | --- |
| **Int CheckToMove()** | int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i,j;  for(i=0; i<4; i++)  {  for(j=0; j<4; j++)  {  if(block[currentBlock][blockRotate][i][j])  {  if((blockY+i>=HEIGHT) || (blockX+j)<0 || (blockX+j>=WIDTH))  return 0;  if(f[blockY+i][blockX+j] != 0)  return 0;  }  }  }  return 1;  } |
| 이 함수는 예비보고서의 pseudo code와 달라지지 않았다.  시간복잡도 : O(BlOCK\_HEIGHT \* BLOCK\_WIDTH)  공간복잡도 : O(1) |
| **Void DrawChange()** | void DrawChange(char f[HEIGHT][WIDTH],int command,int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i,j;  int tempBlock, tempUp, x, y;  tempBlock = currentBlock;  tempUp = blockRotate;  y = blockY; x = blockX;  switch(command)  {  case KEY\_UP:  tempUp = (tempUp+3)%4;  break;  case KEY\_DOWN:  y--;  break;  case KEY\_LEFT:  x++;  break;  case KEY\_RIGHT:  x--;  break;  }  for(i=0; i<4; i++)  {  for(j=0; j<4; j++)  {  if(block[tempBlock][tempUp][i][j] == 1)  {  if(y+i>=0)  {  move(i+y+1, j+x+1);  printw(".");  }  }  }  }  //DrawBlock(blockY,blockX,currentBlock,blockRotate,' ');  DrawBlockWithFeatures(blockY, blockX, currentBlock, blockRotate);  move(HEIGHT, WIDTH+10);  } |
| 예비보고서에서는 switch부분에서 이전 좌표 값이 아닌 다른 값을 가져왔었다.  시간복잡도 : O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)  공간복잡도 : O(1) |
| **Void BlockDown()** | void BlockDown(int sig){  // user code  timed\_out = 0;  int check;  check = CheckToMove(field, nextBlock[0], blockRotate, blockY+1, blockX);  if(check ==1)  {  blockY = blockY + 1;  DrawChange(field, KEY\_DOWN, nextBlock[0], blockRotate, blockY, blockX);  }  else if(check !=1 && blockY == -1)  gameOver = 1;  else if(check !=1 && blockY != -1)  {  AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);  //DrawNextBlock(nextBlock);  score = score + DeleteLine(field);  nextBlock[0] = nextBlock[1];  nextBlock[1] = nextBlock[2];  nextBlock[2] = rand()%7;  blockRotate=0;  blockY=-1;  blockX=WIDTH/2-2;  DrawBlockWithFeatures(blockY, blockX, nextBlock[0], blockRotate);  DrawNextBlock(nextBlock);  PrintScore(score);  }  } |
| 예비보고서에서는 이 함수에서 스코어 갱신, 다음블록을 현재블록으로 바꾸기, 랜덤으로 선정하기, 스코어 구하기, 필드 그리기 등 프로그램전반의 초기화나 설정들은 해주지 않았었다.  시간복잡도 : O(1)  공간복잡도 : O(1) |
| **Void AddBlockToField()** | void AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i,j;  int touch = 0;  for(i=0 ; i<4; i++)  {  for(j=0; j<4; j++)  {  if(block[currentBlock][blockRotate][i][j] == 1)  {  if(f[blockY+i+1][blockX+j] || blockY+i+1>=HEIGHT)  touch=touch+1;  f[blockY+i][blockX+j] = 1;  }  }  }  score = score + (touch\*10);  } |
| 이 함수는 예비보고서와 코드 내용이 같았다.  시간 복잡도 : O(BLOCK\_HEIGHT\*BLOCK\_WIDTH)  공간 복잡도 : O(1) |
| **Int DeleteLine()** | int DeleteLine(char f[HEIGHT][WIDTH]){  // user code  int i,j, k, temp;  int delete\_score =0;  for(i=HEIGHT-1; i>=0; i--)  {  temp=0;  for(j=0; j<WIDTH; j++)  {  if(field[i][j] !=1)  {  temp=1;  break;  }  }  if(temp !=1)  {  for(k=i; k>0; k--)  {  for(j=0; j<WIDTH; j++)  f[k][j] = f[k-1][j];  }  DrawField();  i++;  delete\_score = delete\_score+1;  }  }  return 100\*delete\_score\*delete\_score;  } |
| 예비보고서에서는 블록이 꽉 차있는 경우 해당 라인을 삭제하는 것만 생각하였으나 실제로는 지워진 라인을 세어 점수도 갱신하고 for문을 통해 지워진 만큼 전체 블록을 내리는 것까지 수행했다.  시간 복잡도 : O(HEIGHT\*(WIDTH + 제거할 라인수 \* WIDTH))  공간 복잡도 : O(1) |

**2. 테트리스 프로젝트 1주차 수제 문제를 해결하기 위한 pseudo code를 기술하고, 작성한 pseudo code의 시간 및 공간 복잡도를 보이시오.**

|  |  |
| --- | --- |
| **void DrawBlock()** | \* 이중 반복문을 통해 tile로 블록을 그린다.  void DrawBlock(int y, int x, int blockID,int blockRotate,char tile){  int i,j;  for(i=0;i<4;i++)  for(j=0;j<4;j++){  if(block[blockID][blockRotate][i][j]==1 && i+y>=0){  move(i+y+1,j+x+1);  attron(A\_REVERSE);  printw("%c",tile);  attroff(A\_REVERSE);  }  }  move(HEIGHT,WIDTH+10);  }  복잡도 : o(1) |
| **void DrawShadow()** | \* CheckToMove함수를 통해 블록이 내려올 수 있는지 여부를 확인하여 가능하면 ‘/’를 통해 그림자를 그린다.  void DrawShadow(int y, int x, int blockID,int blockRotate){  // user code  int check;  int i;  for(i=y; i<HEIGHT; i++)  {  check=CheckToMove(field, blockID, blockRotate, i, x);  if(check==0)  break;  }  DrawBlock(i-1, x, blockID, blockRotate, '/');  }  복잡도 o(1) |
| **void DrawBlockWithFeatures()** | \* DrawBlock(), DrawShadow() 함수를 호출하는 함수로, 기존의 DrawBlock() 함수의 위치에 삽입하여 움직임이 갱신될 때마다 현재 블록과 그림자를 함께 그린다.  void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate)  {  DrawField();  DrawShadow(y, x, blockID, blockRotate);  DrawBlock(y, x, blockID, blockRotate, ' ');  }  복잡도 X |
| **void InitTetris()** | \* nextBlock[2]=rand()%7 를 원래 함수에 추가하여 다음 블록 미리보기를 위한 배열의 크기를 증가시키고, 난수를 생성한다.  복잡도 X |
| **void DrawNextBlock()** | \* 함수는 위에 그린 블록의 초기 위치를 변경하여 2번째 next 블록을 그리는 과정을 추가한다.  //추가한 부분만 발췌//  for (i=0; i<4; i++)  {  move(10+i, WIDTH+13);  for(j=0; j<4; j++)  {  if(block[nextBlock[2]][0][i][j] == 1)  {  attron(A\_REVERSE);  printw(" ");  attroff(A\_REVERSE);  }  else  printw(" ");  복잡도 : O(1) |
| **void BlockDown()** | nextBlock[2]=rand()%7;  DrawBlockWithFeatures(blockY, blockX, nextBlock[0], blockRotate);  두 줄을 추가  복잡도 : O(1) |