8주차 결과보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

1. 실습 시간에 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
(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)

i, j에 의한 연산 회수가 각각 상수 회수만큼 실행되므로 **시간 복잡도는 O(1)**이고, 블록의 크기만큼 공간을 사용하므로 **공간 복잡도는 O(1)**이다.

1) 기존에 작성했던 pseudo code와 달리, 블록이 실제로 존재하는 부분에 대해서만 이동 여부를 판단하도록 했다(2중 조건문으로 변경), 기존 코드에는 테트리스 **게임의 마지막 세 번째 줄에서 블록별 모양 차이에 의해 게임이 화면보다 훨씬 빨리 종료되는 버그가 있었다.**

이를 해결하기 위해 블록이 처음 생성되는 blockY 값을 감소시키는 대신, 블록이 생성되자마자 gameOver가 발생하지 않도록 ty가 0 이하인 조건문을 없애는 한편, field[ty][tx]에서 경계를 넘는 접근이 발생하지 않도록 ty가 -1 이하라면 continue하도록 했다.

2) 블록 초기 위치(blockY)를 -3으로 감소시키면서 **테트리스 블록이 생성되자마자 왼쪽 키나 오른쪽 키를 빠르게 눌러 필드 바깥으로 이동시키면 gameOver가 발생하는 버그가 있었다.** 특정한 블록의 경우 블록 초기 생성 위치가 필드보다 위에서 잡히는데, 이 때 왼쪽, 오른쪽 이동에 대해 CheckToMove()가 제대로 작동되지 않아 blockX가 필드 경계 밖으로 빠져나갔기 때문이다. 문제를 해결하기 위해 블록이 필드 바깥에 있을 때 tx가 경계 밖에 있다면 FALSE를 반환하는 조건문을 추가해 버그를 해결했다.

```
int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX) {
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      int ty = blockY + i;
      int tx = blockX + j;
      if (block[currentBlock][blockRotate][i][j] == 1) {
        if (ty <= -1) {
          if (tx >= 0 && tx < WIDTH)
            continue;
          else
            return FALSE;
        }
        if (ty >= HEIGHT || tx < 0 || tx >= WIDTH) return FALSE;
        if (field[ty][tx] == 1) return FALSE;
      }
    }
  }
  return TRUE;
}
```

void DrawChange
(char f[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX)

i, j에 의한 연산이 각각 상수 회수만큼 실행되며, 이외에는 단순 조건문들이므로 역시 **시간 복잡도는 O(1)**이다. 블록의 크기만큼 메모리를 사용하므로 **공간 복잡도 역시 O(1)**이다.

의사코드에선 DrawField() 명령어만으로 이전 블록을 삭제하고자 했지만, 생각대로 실행되지 않아 ppt를 참고하여 제시된 순서에 맞게 코드를 다시 작성했다. 이전 블록 정보를 찾고, 블록을 삭제한 뒤, 커서를 이동하고, 현재 블록 정보를 복구한 뒤 새로운 블록을 그리는 과정을 그대로 구현했다.

```
void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int
blockRotate, int blockY, int blockX) {
  // 1. 이전 블록 정보를 찾는다. ProcessCommand의 switch문을 참조할 것
  // 2. 이전 블록 정보를 지운다. DrawBlock함수 참조할 것.
  // 3. 새로운 블록 정보를 그린다.

  if (command == KEY_UP) blockRotate = (blockRotate + 3) % 4;
  if (command == KEY_DOWN) blockY--;
  if (command == KEY_LEFT) blockX++;
  if (command == KEY_RIGHT) blockX--;

  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++) {
      if (block[currentBlock][blockRotate][i][j] == 1 && blockY + i >= 0) {
        move(i + blockY + 1, j + blockX + 1);
        printw(".");
      }
    }

  move(HEIGHT, WIDTH + 10);

  if (command == KEY_UP) blockRotate = (blockRotate + 1) % 4;
  if (command == KEY_DOWN) blockY++;
  if (command == KEY_LEFT) blockX--;
  if (command == KEY_RIGHT) blockX++;

  DrawBlock(blockY, blockX, currentBlock, blockRotate, ' ');
}
```

void BlockDown(int sig)

안쪽에 존재하는 각 함수별로 시간 복잡도를 살펴보면, CheckToMove()는 O(1), DrawChange()도 O(1), PrintScore()도 O(1), DrawNextBlock(), AddBlockToField()도 O(1)이다. 단 DrawField()는 O(HEIGHT * WIDTH), DeleteLine()은 O(HEIGHT^2 * WIDTH)의 시간 복잡도가 발생한다. 따라서 BlockDown()의 시간 복잡도는 O(HEIGHT^2 * WIDTH)이다. 이 때 HEIGHT, WIDTH는 field의 크기이다. 공간 복잡도 역시 필드 크기만큼 메모리에 접근하는 것이 최대값이므로 O(HEIGHT * WIDTH)이다.

CheckToMove()에서 설명했던 버그를 해결하기 위해 blockY == -3으로 조건문을 변경했으며, timed_out = 0을 추가해 함수 종료 시마다 timed_out 변수를 갱신하도록 했다. 또한 첫 줄의 조건문을 blockY + 1이 아닌 blockY로 작성해 둔 것을 고쳤다.

```
void BlockDown(int sig) {
  if (CheckToMove(field, nextBlock[0], blockRotate, blockY + 1, blockX) == TRUE) {
    blockY++;
    DrawChange(field, KEY_DOWN, nextBlock[0], blockRotate, blockY, blockX);
  } else {
    if (blockY == -3) {
      gameOver = TRUE;
    } else {
      AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);
      score += DeleteLine(field);

      nextBlock[0] = nextBlock[1];
      nextBlock[1] = rand() % 7;
      blockRotate = 0;
      blockY = -3;
      blockX = WIDTH / 2 - 2;

      DrawNextBlock(nextBlock);
      PrintScore(score);
      DrawField();
    }
  }
  timed_out = 0;
}
```

void AddBlockToField
(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)

시간 복잡도는 i, j가 상수 회수만큼 실행되므로 O(1)이며, 공간 복잡도 역시 블록 크기만큼이므로 O(1)이다.

의사 코드에 비해 크게 차이가 없으며, 이중 조건문으로 변경하고, ty가 0 이하일 때의 조건문만 삭제했다. 블록이 필드에 추가될 수 있는지, 없는지는 이미 CheckToMove()에서 확인했기 때문에 삭제에 무리가 없다.

```
void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate,
int blockY, int blockX) {
  // Block이 추가된 영역의 필드값을 바꾼다.
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      int ty = blockY + i;
      int tx = blockX + j;
      if (block[currentBlock][blockRotate][i][j] == 1) {
        if (ty >= HEIGHT || tx < 0 || tx >= WIDTH) continue;
        field[ty][tx] = 1;
      }
    }
  }
}
```

int DeleteLine(char f[HEIGHT][WIDTH])

field를 행마다 탐색하고, 행이 꽉 차 있는 경우 모든 행을 한 칸씩 내리는 작업을 수행한다. 최악의 경우는 모든 행이 꽉 차 있는 경우이다. 매 line마다 자기보다 윗 line에 있는 값을 아래로 내리는 작업을 수행하게 되므로,

HEIGHT * WIDTH * HEIGHT
(행의 개수) * (행의 윗쪽에 있는 필드의 칸)

만큼의 작업이 수행된다. 행이 꽉 차 있는지 확인하는 연산이 추가로 들지만, 해당 연산은 WIDTH * HEIGHT만큼만 발생하므로 생략된다. 따라서 시간 복잡도는 O(WIDTH * HEIGHT^2)이다.
field 크기만큼 메모리에 접근하므로 공간 복잡도는 O(HEIGHT * WIDTH)이다.
의사 코드와 동일하게 작성했다.

```
int DeleteLine(char field[HEIGHT][WIDTH]) {
  // 1. 필드를 탐색하여, 꽉 찬 구간이 있는지 탐색한다.
  // 2. 꽉 찬 구간이 있으면 해당 구간을 지운다. 즉, 해당 구간으로 필드값을 한칸씩 내린다.
  int del_line = 0;
  for (int i = 0; i < HEIGHT; i++) {
    int full_line = TRUE;
    for (int j = 0; j < WIDTH; j++) {
      if (field[i][j] == 0) {
        full_line = FALSE;
        break;
      }
    }
    if (full_line == TRUE) {
      del_line++;
      for (int p = i; p >= 0; p--) {
        for (int q = 0; q < WIDTH; q++) {
          field[p][q] = field[p - 1][q];
        }
      }
      for (int q = 0; q < WIDTH; q++)
        field[0][q] = 0;
      i--;
    }
  }

  return del_line * del_line * 100;
}
```

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

void InitTetris()
nextBlock[2]까지 블록을 생성하도록 수정하는 것 외엔 변경 사항이 없다.

void BlockDown(int sig)
nextBlock[2]까지 블록을 생성하도록 수정하는 것 외엔 변경 사항이 없다.

void DrawBlock((int y, int, x, int blockID, int blockRotate, char tile)
ppt에는 변경할 함수로 나타나 있지만 특별히 변경할 사항이 없었다. **이전과 동일하게 시간, 공간 복잡도 모두 O(1)**이다.

void DrawNextBlock(int *nextBlock)
특별한 변경 사항 없이 기존의 NextBlock 반복문을 복사했다. 각 반복문이 블록 크기(4*4)만큼 2차원 배열에 접근하므로 각 반복문의 시간 복잡도는 O(1)이고, 다음 블록의 개수에 따라 시간 복잡도가 발생하므로 **시간 복잡도는 O(BLOCK_NUM)**이다. **공간 복잡도는 O(1)**이다.

void DrawShadow(int y, int, x, int blockID, int blockRotate)
필드 높이만큼 내려가면서 시간 복잡도가 O(1)인 함수 CheckToMove()를 실행하므로 **시간 복잡도는 O(HEIGHT)**이며, 공간 복잡도는 O(1)이다. y++ while (CheckToMove(field, nextBlock[0], blockRotate, y, x))   y++ DrawBlock(y - 1, x, blockID, blockRotate, '/')

void DrawBlockWithFeatures(int y, int, x, int blockID, int blockRotate)
단순히 DrawBlock()과 DrawShadow()를 호출하는 함수로 사용된다. DrawBlock과 DrawShadow 양쪽의 시, 공간 복잡도가 모두 O(1)이므로 DrawBlockWithFeatures()의 **시간 복잡도, 공간 복잡도는 O(1)**이다. DrawShadow(y, x, blockID, blockRotate) DrawBlock(y, x, blockID, blockRotate, ' ')

void AddBlockToField
(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)

블록의 가장 아랫부분일 때, 필드에 닿은 부분만큼 점수를 증가시킨다. 블록의 4 * 4 공간 중 가장 아래에 존재하는 블록이거나, 블록의 아랫부분에 더 이상 공간이 없다면 가장 아랫부분이다. 또한, 닿은 공간의 아래가 필드의 아래쪽 바깥이거나, 블록이 존재한다면 필드에 닿은 부분이다.

시간 복잡도, 공간 복잡도는 모두 블록 크기에 따르므로, O(1)이다.

```
for i = 0 to 3
  for j = 0 to 3
    ty = blockY + i, tx = blockX + j
    if (block[currentBlock][blockRotate][i][j] == 1)
      if (ty >= HEIGHT || tx < 0 || tx >= WIDTH) continue;
      field[ty][tx] = 1;
      if (i == 3 || block[currentBlock][blockRotate][i + 1][j] == 0)
        if (ty + 1 >= HEIGHT || field[ty+1][tx] == 1) score += 10
```