CSE3013 (컴퓨터공학 설계 및 실험 I)
PRJ-1 테트리스 프로젝트 1주차 결과 보고서

서강대학교 컴퓨터공학과 박수현 (20181634)

서강대학교 컴퓨터공학과

1 목적

1주차 구현 내용과 작성했던 의사 코드와 실제로 어떻게 다른지 기술하고 복잡도를 보인다. 1주차 과제 내용에 대한 의사 코드와 복잡도를 보인다.

2 의사 코드 비교

의사 코드를 그대로 C 코드로 변환했으며, 실제 코드는 의사 코드와 거의 차이가 없었다. 아래는 의사 코드와 실제 코드의 비교이다.

CHECKTOMOVE : 명령에 따라 블럭을 이동할 수 있는지 판단한다. 블럭 배열 크기인 4개의 행과 4개의 열마다 $\mathcal{O}(1)$의 작업을 수행하므로, 시간 복잡도는 $\mathcal{O}(1)$이다.

CHECKTOMOVE($f, currentBlock, blockRotate, blockY, blockX$)

```
1   for i = 0 to 3
2       for j = 0 to 3
3           if block[currentBlock][blockRotate][i][j] == 1
4               x = blockX + j
5               y = blockY + i
6               if (0 ≤ x < WIDTH and 0 ≤ y < HEIGHT) ≠ TRUE
7                   return FALSE
8               if f[y][x] == 1
9                   return FALSE
10  return TRUE
```

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

```
248     for (int i = 0; i < 4; i++) {
249         for (int j = 0; j < 4; j++) {
250             if (block[currentBlock][blockRotate][j][i]) {
251                 int x = blockX + i;
252                 int y = blockY + j;
253                 if (!(0 <= x && x < WIDTH && 0 <= y && y < HEIGHT))
254                     return 0;
255                 if (f[y][x])
256                     return 0;
257             }
258         }
259     }
260     return 1;
261 }
```

DRAWCHANGE : 명령에 의해 바뀐 부분만 필드에 업데이트한다. DRAWFIELD에 $\mathcal{O}(WIDTH \times HEIGHT)$ 만큼, DRAWBLOCK에 $\mathcal{O}(1)$만큼의 시간이 걸리므로 이 함수의 시간 복잡도는 $\mathcal{O}(WIDTH \times HEIGHT)$ 이다.

DRAWCHANGE$(f, command, currentBlock, blockRotate, blockY, blockX)$

1 // Erase current block
2 DRAWFIELD()
3 // Draw new block
4 DRAWBLOCK$(blockY, blockX, currentBlock, blockRotate, ' ')$

```
263 void DrawChange(char f[HEIGHT][WIDTH], int command, int currentBlock,
264                 int blockRotate, int blockY, int blockX) {
265     DrawField();
266     DrawBlock(blockY, blockX, currentBlock, blockRotate, ' ');
267 }
```

BLOCKDOWN : 블럭을 아래로 내린다. 더 이상 내릴 수 없을 경우 현재 블럭을 필드에 고정시키고 다음 블럭을 사용한다. 시간 복잡도는 CHECKTOMOVE와 ADDBLOCKTOFIELD에 각각 $\mathcal{O}(1)$, DRAWCHANGE와 DELETELINE과 DRAWFIELD에 각각 $\mathcal{O}(WIDTH \times HEIGHT)$이 걸리므로 $\mathcal{O}(WIDTH \times HEIGHT)$이다.

BLOCKDOWN(sig)

```
1  if CHECKTOMOVE(f, currentBlock, blockRotate, blockY + 1, blockX)
2      blockY = blockY + 1
3      DRAWCHANGE(f, command, currentBlock, blockRotate, blockY, blockX)
4  else
5      if blockY == 1
6          gameOver = TRUE
7      else
8          ADDBLOCKTOFIELD(f, currentBlock, blockRotate, blockY, blockX)
9          score = score + DELETELINE(f)
10         // Generate new block
11         nextBlock[0] = nextBlock[1]
12         nextBlock[1] = (Random integer in 0 .. 6)
13         DRAWNEXTBLOCK(nextBlock)
14         blockY = −1, blockX = (Center of field)
15         DRAWFIELD()
16         PRINTSCORE(score)
```

```
269 void BlockDown(int sig) {
270     if (CheckToMove(field, nextBlock[0], blockRotate, blockY + 1, blockX)) {
271         blockY++;
272         DrawChange(field, KEY_DOWN, nextBlock[0], blockRotate, blockY, blockX);
273     } else {
274         if (blockY == 1) {
275             gameOver = 1;
276         } else {
277             AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);
278             score += DeleteLine(field);
279 
280             nextBlock[0] = nextBlock[1];
281             nextBlock[1] = rand() % 7;
282             DrawNextBlock(nextBlock);
283 
284             blockY = -1, blockX = WIDTH / 2 - 2;
285             DrawField();
```

```
286                 PrintScore(score);
287             }
288         }
289         timed_out = 0;
290     }
```

의사 코드에서 $nextBlock[1] =$ (Random integer in 0 .. 6)이 `nextBlock[1] = rand() % 7`로, $blockX =$ (Center of field)이 `blockX = WIDTH / 2 - 2`로 구현되었다.

AddBlockToField : 필드의 주어진 좌표에 현재 블럭을 고정시킨다. 블럭 배열 크기인 4개의 행과 4개의 열마다 $\mathcal{O}(1)$의 작업을 수행하므로, 시간 복잡도는 $\mathcal{O}(1)$이다.

AddBlockToField($f, currentBlock, blockRotate, blockY, blockX$)

```
1  for i = 0 to 3
2      for j = 0 to 3
3          if block[currentBlock][blockRotate][i][j] == 1
4              f[blockY + i][blockX + j] = 1
5  return TRUE
```

```
292 void AddBlockToField(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate,
293                     int blockY, int blockX) {
294     for (int i = 0; i < 4; i++) {
295         for (int j = 0; j < 4; j++) {
296             if (block[currentBlock][blockRotate][i][j]) {
297                 f[blockY + i][blockX + j] = 1;
298             }
299         }
300     }
301 }
```

DELETELINE : 꽉 찬 줄이 있는지 체크하고, 있을 경우 줄을 지우고 스코어를 증가시킨다. 모든 열마다 모든 칸이 차 있는지 체크하므로 시간 복잡도는 $\mathcal{O}(WIDTH \times HEIGHT)$이다.

DELETELINE(f)

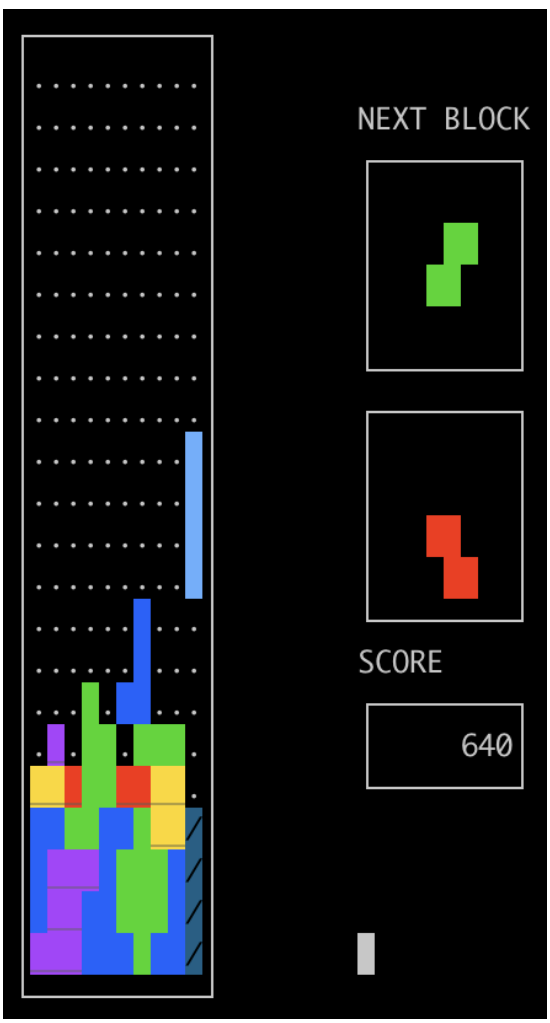
```
1  erased = 0
2  for i = 0 to HEIGHT − 1
3      flag = TRUE
4      for j = 0 to WIDTH − 1
5          if f[i][j] == 0
6              flag = FALSE
7      if flag == TRUE
8          erased = erased + 1
9          for y = i downto 1
10             for x = 0 to WIDTH − 1
11                 f[y][x] = f[y − 1][x]
12         for x = 0 to WIDTH − 1
13             f[0][x] = 0
14         i = i − 1
15 return 100 × erased^2
```

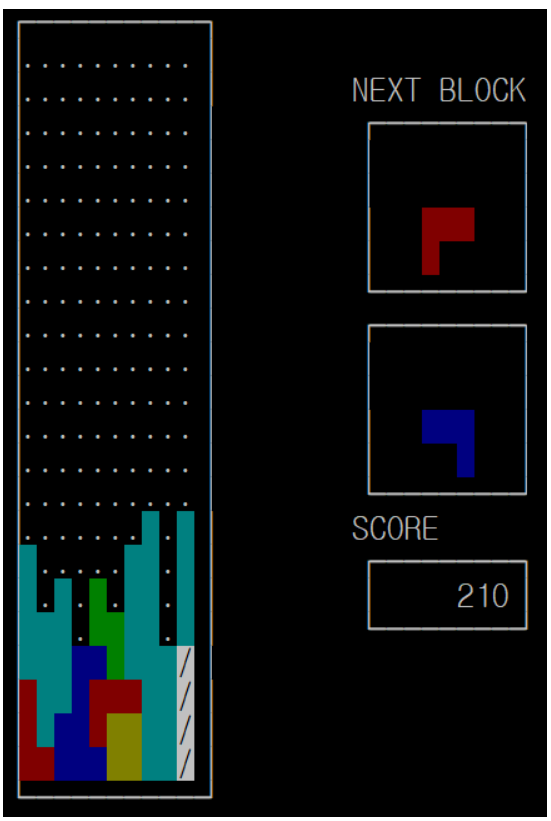
```
303 int DeleteLine(char f[HEIGHT][WIDTH]) {
304     int erased = 0;
305     for (int i = 0; i < HEIGHT; i++) {
306         int flag = 1;
307         for (int j = 0; j < WIDTH; j++) {
308             if (!f[i][j])
309                 flag = 0;
310         }
311         if (flag) {
312             erased++;
313             for (int y = i; y >= 1; y--) {
314                 for (int x = 0; x < WIDTH; x++) {
315                     f[y][x] = f[y - 1][x];
316                 }
317             }
318             for (int x = 0; x < WIDTH; x++) {
319                 f[0][x] = 0;
320             }
321             i--;
322         }
```

```
323         }
324
325         return 100 * erased * erased;
326     }
```

3 과제

프로젝트 요구사항에 추가로 필드가 색상 정보를 담을 수 있도록 자료구조를 다소 수정하였다. 필드 각각의 칸에 담겨 있는 수가 0이면 비어 있음을 뜻하고, 0이 아니면 차 있음을 뜻하며, 0이 아닐 경우 렌더해야 하는 색상을 의미하도록 정의를 바꾸었다.





256 색상 렌더를 지원하는 터미널에서는 256 색상을 사용해 렌더했고, 그렇지 않은 터미널에서는 8 색상만을 이용해 렌더했다.

3.1 그림자 렌더

DRAWBLOCK : 블럭을 그린다. 시간 복잡도는 $\mathcal{O}(1)$이다.

DRAWBLOCK$(y, x, blockID, blockRotate, tile, isShadow)$

1 **for** $i = 0$ **to** 3
2 **for** $j = 0$ **to** 3
3 **if** $block[blockID][blockRotate][i][j]$ == 1 and $i + y \geq 0$
4 Move cursor to $(i + y + 1, j + x + 1)$
5 Print character
6 Move cursor to $(HEIGHT, WIDTH + 10)$

```
190 void DrawBlock(int y, int x, int blockID, int blockRotate, char tile, int isShadow) {
191     int i, j;
192     for (i = 0; i < 4; i++) {
193         for (j = 0; j < 4; j++) {
194             if (block[blockID][blockRotate][i][j] == 1 && i + y >= 0) {
195                 move(i + y + 1, j + x + 1);
196                 if (isShadow) {
197                     attron(A_REVERSE | COLOR_PAIR(ShadowColor[blockID]));
198                     printw("%c", tile);
199                     attroff(A_REVERSE | COLOR_PAIR(ShadowColor[blockID]));
200                 } else {
201                     attron(A_REVERSE | COLOR_PAIR(Color[blockID]));
202                     printw("%c", tile);
203                     attroff(A_REVERSE | COLOR_PAIR(Color[blockID]));
204                 }
205             }
206         }
207     }
208
209     move(HEIGHT, WIDTH + 10);
210 }
```

GHOSTY : 현재 블럭이 그대로 떨어질 경우 도달하는 블럭의 y좌표를 계산한다. 시간 복잡도는 $\mathcal{O}(HEIGHT)$이다.

GHOSTY$(y, x, blockID, blockRotate)$

```
1  while CHECKTOMOVE(field, nextBlock[0], blockRotate, y + 1, x)
2      y = y + 1
3  return y
```

```
292 int GhostY(int y, int x, int blockID, int blockRotate) {
293     while (CheckToMove(field, nextBlock[0], blockRotate, y + 1, x)) y++;
294     return y;
295 }
```

DRAWSHADOW : 그림자를 그린다. 시간 복잡도는 $\mathcal{O}(HEIGHT)$이다.

DRAWBLOCK$(y, x, blockID, blockRotate, tile, isShadow)$

```
1  y = GHOSTY(y, x, blockID, blockRotate)
2  DRAWBLOCK(y, x, blockID, blockRotate, '/', 1)
```

```
378 void DrawShadow(int y, int x, int blockID, int blockRotate) {
379         y = GhostY(y, x, blockID, blockRotate);
380         DrawBlock(y, x, blockID, blockRotate, '/', 1);
381 }
```

DRAWBLOCKWITHFEATURES : 블럭과 그림자를 그린다. 시간 복잡도는 $\mathcal{O}(HEIGHT)$이다.

DRAWBLOCKWITHFEATURES$(y, x, blockID, blockRotate)$

```
1  DRAWSHADOW(y, x, blockID, blockRotate)
2  DRAWBLOCK(y, x, blockID, blockRotate, ' ', 0)
```

```
383 void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate) {
384         DrawShadow(y, x, blockID, blockRotate);
385     DrawBlock(y, x, blockID, blockRotate, ' ', 0);
386 }
```

3.2 다음 블록 슬롯 추가

BLOCK_NUM의 값을 바꿀 경우 칸이 늘어나게 설계하였다. NEXT 슬롯의 높이는 6칸이므로, 원래 SCORE 슬롯이 Y 좌표 10에 그려졌다면 수정된 이후에는 $-2+6\times BLOCK_NUM$이 된다.

INITTETRIS - 다음 블럭을 초기화하는 부분이 아래와 같이 수정되었다.

```
54      for (int i = 0; i < BLOCK_NUM; i++) {
55          nextBlock[i] = rand() % 7;
56      }
```

DRAWNEXTBLOCK - 다음과 같이 수정되었다. 시간 복잡도는 $\mathcal{O}(BLOCK_NUM)$이다.

```
174 void DrawNextBlock(int *nextBlock) {
175     for (int b = 1; b < BLOCK_NUM; b++) {
176         for (int i = 0; i < 4; i++) {
177             move(-2 + i + 6 * b, WIDTH + 13);
178             for (int j = 0; j < 4; j++) {
179                 if (block[nextBlock[b]][0][i][j] == 1) {
180                     attron(A_REVERSE | COLOR_PAIR(Color[nextBlock[b]]));
181                     printw(" ");
182                     attroff(A_REVERSE | COLOR_PAIR(Color[nextBlock[b]]));
183                 } else
184                     printw(" ");
185             }
186         }
187     }
188 }
```

BLOCKDOWN - 다음 블럭 목록을 업데이트하는 부분이 아래와 같이 수정되었다.

```
315             for (int i = 0; i < BLOCK_NUM - 1; i++) {
316                 nextBlock[i] = nextBlock[i + 1];
317             }
```

3.3 점수 계산 방법 변경

ADDBLOCKTOFIELD - 다음과 같이 수정되었다. 시간 복잡도는 $\mathcal{O}(1)$이다.

```
329 int AddBlockToField(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate,
330                     int blockY, int blockX) {
331     int adjacentCount = 0;
332     for (int i = 0; i < 4; i++) {
333         for (int j = 0; j < 4; j++) {
334             if (!block[currentBlock][blockRotate][i][j])
335                 continue;
```

```
336
337             int ny = blockY + i, nx = blockX + j;
338             if (0 > ny || ny >= HEIGHT)
339                 continue;
340             if (0 > nx || nx >= WIDTH)
341                 continue;
342
343             f[ny][nx] = Color[currentBlock];
344             if (ny == HEIGHT - 1)
345                 adjacentCount++;
346             else if (f[ny + 1][nx] != 0)
347                 adjacentCount++;
348         }
349     }
350     return adjacentCount * 10;
351 }
```

BlockDown - 점수가 추가되는 부분이 다음과 같이 수정되었다.

```
308         score +=
309             AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);
310         if (blockY == -1) {
311             gameOver = true;
312         } else {
313             score += DeleteLine(field);
314
315             for (int i = 0; i < BLOCK_NUM - 1; i++) {
316                 nextBlock[i] = nextBlock[i + 1];
317             }
318             nextBlock[BLOCK_NUM - 1] = rand() % 7;
319             DrawNextBlock(nextBlock);
320
321             blockY = -1, blockX = WIDTH / 2 - 2;
322             DrawField();
323             PrintScore(score);
324         }
```