테트리스 프로젝트

20190808 방지혁

목차

1. 프로젝트 목표 및 실험 환경

- 본 프로젝트는 수업에서 실습한 기존의 테트리스 추천 알고리즘을 개선하자!
  - Beam Search 알고리즘을 통한 최적화
  - 필드 저장 방식 변환 -> 평가 정밀화
  - 필드 평가 함수 개선: 구멍 개수 세기, 울퉁불퉁 정도 계산, 높이 차이 계산, 골짜기 계산, 채운 라인 계산
- 실행 환경
  - macOS에서 개발
  - 기존과 똑같이 ncurses 라이브러리가 필요함
    - gcc tetris.c -lncurses -o tetris -> 컴파일
    - ./tetris로 실행

2. 각 변수 설명

게임 필드 관련

- char field[HEIGHT][WIDTH];
- **블록 정보**
- Int nextBlock[BLOCK_NUM];.
- **현재 블록 상태 관련**
- Int blockRotate; - 회전 수 저장
- Int blockY; - y 좌표 저장
- Int blockX; - x 좌표 저장

게임 상태

- Int gameover; - 게임 종료 여부 저장, 0이면 아직 진행 중, 1이면 게임 종료된 상태
- Int score; - 현재 게임 점수 저장 변수
- Int timed_out; - 타이머 타임아웃 플래그 변수
- **랭킹 관련**
- Node *head; - 내림차순으로 랭킹 정보를 저장하는 링크드 리스트의 헤드 포인터
- Int score_number; - 랭킹 정보에 등록된 총 랭킹의 수

추천 알고리즘 관련

- RecNode* recRoot; - 트리의 루트 포인터
- Int recommendR; - 추천 회전 횟수
- Int recommendY; - 추천 y 좌표
- Int recommendX; - 추천 x 좌표

3. 각 변수 설명 - 구조체

```
typedef struct _RecNode {
  int level; // 추천 트리의 level(or depth)
  int accumulatedScore; // 누적된 점수
  char recField[HEIGHT][WIDTH]; // 필드 상태
  // tree에서 children 노드들을 가리키는 포인터 배열
  struct _RecNode **child;
  // 블록 위치 정보
  int blockX;
  int blockY;
  int blockRotate;
} RecNode;
```

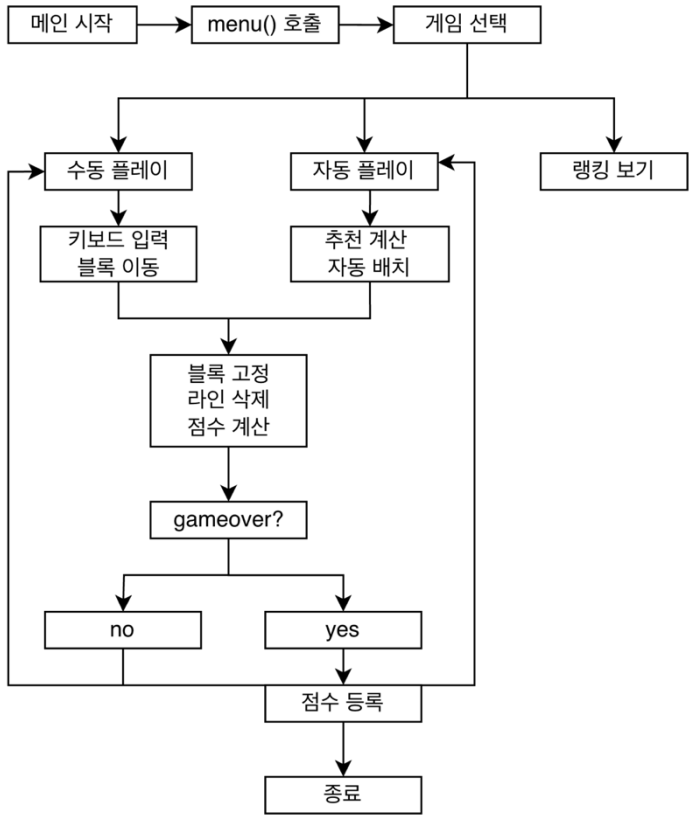
```
typedef struct Node {
  char name[NAMELEN]; // 플레이어 이름 참조
  int score; // 플레이어 점수 저장
  struct Node* next; // 다음 노드를 가리키는 포인터
} Node;
```

3. 각 함수 설명

- int main()
- void InitTetris()
- Int CheckToMove()
- Int AddBlockToField();
- Int DeleteLine();
- Void BlockDown();
- Int evaluateField();
- Int modifiedRecommend();
- Int compareRecNodes();
- Void freeRecNodes();
- Void play();
- Void recommendedPlay();
- Void createRankList();
- Void writeRankFile();
- Void newRank();
- Void insertNode();
- Void rank();

노란색은 추후 설명 예정

4. 프로젝트 전체 플로우 차트



5. 핵심 알고리즘 설명- evaluateField();

```
int evaluateField(char f[HEIGHT][WIDTH]) {
  int holes = 0; // 구멍 수
  int heightColumns[WIDTH] = {0}; // 각 열 높이 저장
  int totalHeight = 0; // 총 높이
  int bumpiness = 0; // 울퉁불퉁 정도
  int wells = 0; // 골짜기 점수
  int maxHeight = 0; // 최대 높이
  int minHeight = HEIGHT; // 최소 높이: 초기값은 최대 높이로 설정
  // 각 열마다 구멍 및 높이 계산
  for (int x = 0; x < WIDTH; x++) {
    int blockFoundFlag = 0; // 블록 발견 플래그
    for (int y = 0; y < HEIGHT; y++) {
      if (f[y][x] != 0) { // 블록이 있음
        blockFoundFlag = 1; // 블록 flag 설정
        if (heightColumns[x] == 0) { // 아직 높이 계산 안됨
          heightColumns[x] = HEIGHT - y;  // 해당 열의 높이 계산
        }
      } else if (blockFoundFlag == 1) { // 블록이 이미 발견된 상태에서 빈 칸
        holes++;  // 구멍 발견
      }
    }
    totalHeight += heightColumns[x]; // 현재 열의 높이를 전체 높이 누적
    if (heightColumns[x] > maxHeight) { // 최대 높이 갱신
      maxHeight = heightColumns[x];
    }
    if (heightColumns[x] < minHeight) { // 최소 높이 갱신
      minHeight = heightColumns[x];
    }
  }
```

```
  // 울퉁불퉁 계산
  for (int i = 0; i < WIDTH - 1; i++) {
    bumpiness += abs(heightColumns[i] - heightColumns[i + 1]);
  }

  // 골짜기 계산
  for (int x = 0; x < WIDTH; x++) {
    int heightleft = HEIGHT;
    int l = x - 1;
    while (l >= 0) { // 왼쪽으로 탐색
      if (heightColumns[l] > heightColumns[x]) {
        heightleft = heightColumns[l]; // 더 높은 열 발견
        break;
      } else if (heightColumns[l] < heightColumns[x]) {
        heightleft = 0; // 더 낮은 열 발견 - 골짜기 아님
        break;
      }
      l--; // 계속 왼쪽으로 이동
    }
    // 오른쪽으로 탐색
    int heightright = HEIGHT;
    int r = x + 1;
    while (r < WIDTH) {
      if (heightColumns[r] > heightColumns[x]) {
        heightright = heightColumns[r]; // 더 높은 열 발견
        break;
      } else if (heightColumns[r] < heightColumns[x]) {
        heightright = 0; // 더 낮은 열 발견 - 골짜기 아님
        break;
      }
      r++; // 계속 오른쪽으로 이동
    }
```

```
    // 둘 중 더 낮은 높이 선택
    int min = 0;
    if (heightleft < heightright)
      min = heightleft;
    else
      min = heightright;
    int depth = 0;
    if (heightColumns[x] < min) {
      depth = min - heightColumns[x];
      if (x == 0 || x == WIDTH - 1) {
        wells += (depth * depth * 10);
      }
      else {
        wells += (depth * depth);
      }
    }
  }
```

5. 핵심 알고리즘 설명- evaluateField();

```
// 높이 차이 계산
int heightDiff = maxHeight - minHeight;
int heightPenalty = 0;

// 최대 높이에 따른 패널티 계산식 다름
if (maxHeight > 18) {
  heightPenalty = 100000;
} else if (maxHeight > 15) {
  heightPenalty = (maxHeight - 15) * 5000;
} else if (maxHeight > 12) {
  heightPenalty = (maxHeight - 12) * 1000;
}
// 완성 가능 라인 보너스
int almostFullBonus = 0;
for (int y = 0; y < HEIGHT; y++) {
  int filled = 0;
  for (int x = 0; x < WIDTH; x++) {
    if (f[y][x] != 0) filled++;
  }
  // 8칸 이상 차있으면 보너스
  if (filled >= WIDTH - 1) {
    almostFullBonus += 3000;
  }
  // 9칸 차있으면 더 큰 보너스
  if (filled >= WIDTH - 1) {
    almostFullBonus += 5000;
  }
}
// 최종 페널티 점수
int penalty = 0;
penalty = holes * 5000 + bumpiness * 100 + totalHeight * 50 + wells * 300 +
          heightPenalty + heightDiff * 800 - almostFullBonus;
return penalty;
```

5. 핵심 알고리즘 설명- beam search (추천 함수)

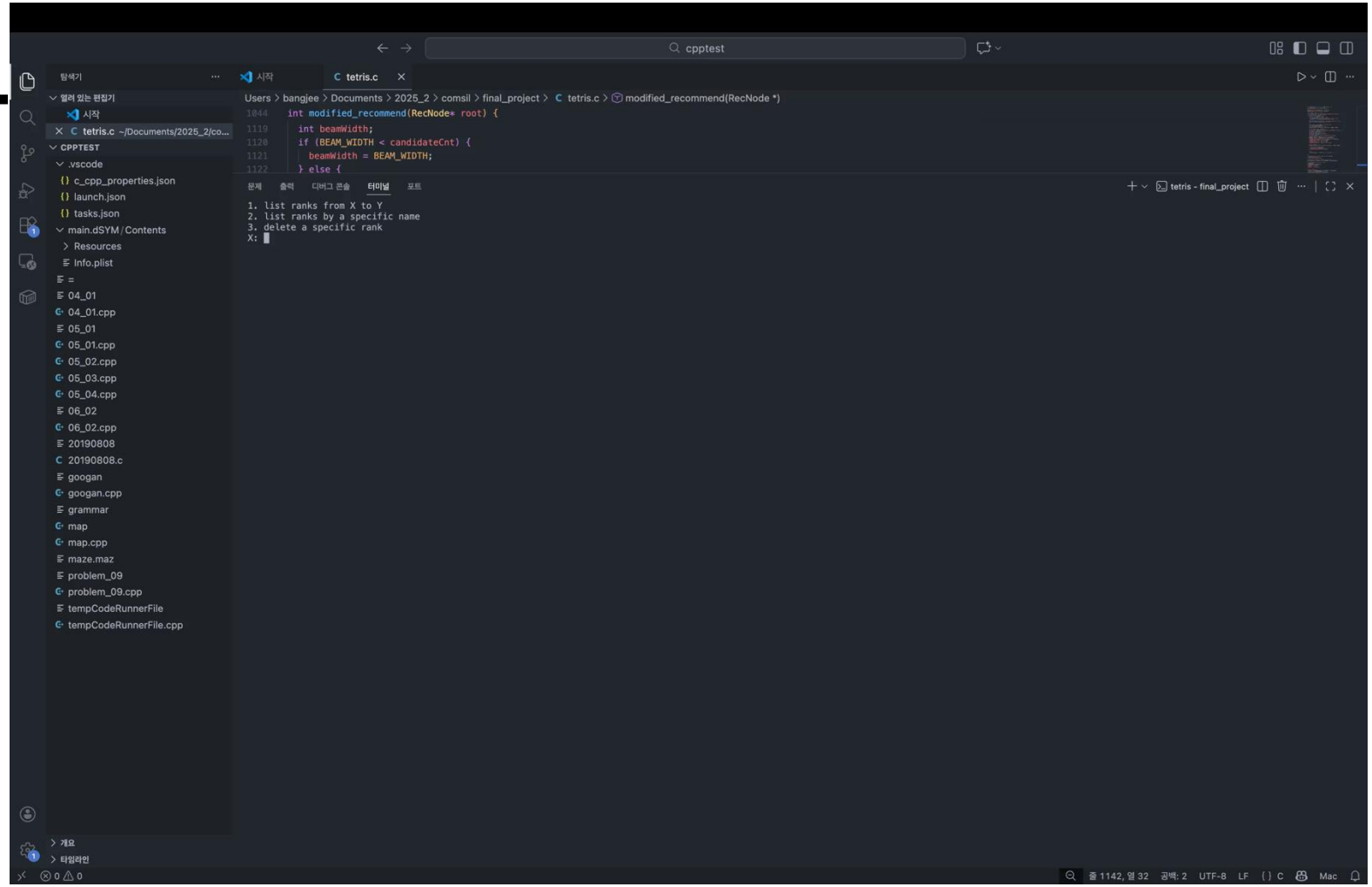
```
// 후보군 정렬: beam search로 점수 기준 내림차순 정렬
qsort(candidates, candidateCnt, sizeof(RecNode*), compareRecNodes);

// 상위 BEAM WIDTH개만 자식 노드로 추가
int beamWidth;
if (BEAM_WIDTH < candidateCnt) {
  beamWidth = BEAM_WIDTH;
} else {
  beamWidth = candidateCnt;
}

// 자식노드 배열 할당
root->child = (RecNode**)malloc(sizeof(RecNode*) * CHILDREN_MAX);
for (int i = 0; i < CHILDREN_MAX; i++) {
  root->child[i] = NULL; // 전부 NULL로 초기화
}
```

```
// 자식 노드들에 대해 재귀적으로 평가
for (int i = 0; i < beamWidth; i++) {
  if (candidates[i] != NULL)
    root->child[i] = candidates[i]; // 자식 노드 연결
  else
    root->child[i] = NULL;
  // 재귀호출
  int childMaxScore = modified_recommend(root->child[i]);
  // 더 크면 최대 점수 갱신
  if (childMaxScore > maxScore) {
    maxScore = childMaxScore;
    // 루트 레벨이면 추천 위치 갱신
    if (currentLevel == 0) {
      recommendX = root->child[i]->blockX; // x좌표
      recommendY = root->child[i]->blockY; // y좌표
      recommendR = root->child[i]->blockRotate; // 회전
    }
  }
}
for (int i = beamWidth; i < candidateCnt; i++) {
  if (candidates[i] != NULL) {
    free(candidates[i]);  // 사용되지 않은 후보군 메모리 해제
  }
}
return maxScore; // 최대 점수 반환
```

6. 실제 구현 영상


cpptest
Users > bangjee > Documents > 2025_2 > comsil > final_project > tetris.c > modified_recommend(RecNode *)
int modified_recommend(RecNode* root) {
int beamWidth;
if (BEAM_WIDTH < candidateCnt) {
beamWidth = BEAM_WIDTH;
} else {
1. list ranks from X to Y
2. list ranks by a specific name
3. delete a specific rank
X:
CPPTEST
.vscode
c_cpp_properties.json
launch.json
tasks.json
main.dSYM / Contents
Resources
Info.plist
04_01.cpp
05_01.cpp
05_02.cpp
05_03.cpp
05_04.cpp
06_02.cpp
20190808.c
googan.cpp
grammar
map.cpp
maze.maz
problem_09.cpp
tempCodeRunnerFile
tempCodeRunnerFile.cpp

7. 창의적 구현에 대한 설명

- Beam Seach - 탐색 후 점수로 정렬을 하고 상위 N개의 노드만 재귀적으로 탐색
- Evaluate Field - 서로 다른 6가지의 특성을 고려 => 가중치 계산

8. 느낀 점 및 개선사항

```
// 모든 가능한 회전에 대해 시도
for (int rotate = 0; rotate < maxRotations[currentBlockID]; rotate++) {
  // 블록을 필드의 모든 x좌표에 대해 시도
  for (int x = -3; x < WIDTH; x++) {
    int y = 0; // 맨 위에서부터 시작
    // 블록이 해당 위치에 놓일 수 있는지 확인
    if (!CheckToMove(root->recField, currentBlockID, rotate, y, x)) {
      continue;  // 해당 위치에 놓을 수 없으면 다음 x좌표로
    }
    while (CheckToMove(root->recField, currentBlockID, rotate, y + 1, x)) {
      y++; // 블록이 더 내려갈 수 있을 때까지 내림
    }
```