

---

<컴퓨터학 실험 I>

## 테트리스 프로젝트 3주차

# 목차

---

- 테트리스 프로젝트 3주차 목표 – 추천 시스템
- 테트리스 프로젝트 3주차 구현 결과
- 테트리스 프로젝트 3주차 구현 프로그램 및 Flow chart
  - 추천 시스템 예제
  - 추천 시스템의 tree 구조의 비효율성 예제
  - 추천 시스템의 tree 구조의 비효율성
  - 비효율성을 해결하기 위한 방법 – pruning tree, data simplification
  - Tree를 이용한 예측 기법의 applications
  - 테트리스 프로그램 전체 흐름과의 관계
  - 추천 시스템 구현을 위한 tree node structure
  - 3주차 구현내용
- 테트리스 프로젝트 3주차 실습 평가
- 테트리스 프로젝트 3주차 숙제
- 테트리스 프로젝트 3주차 결과보고서

# 테트리스 프로젝트 3주차 목표

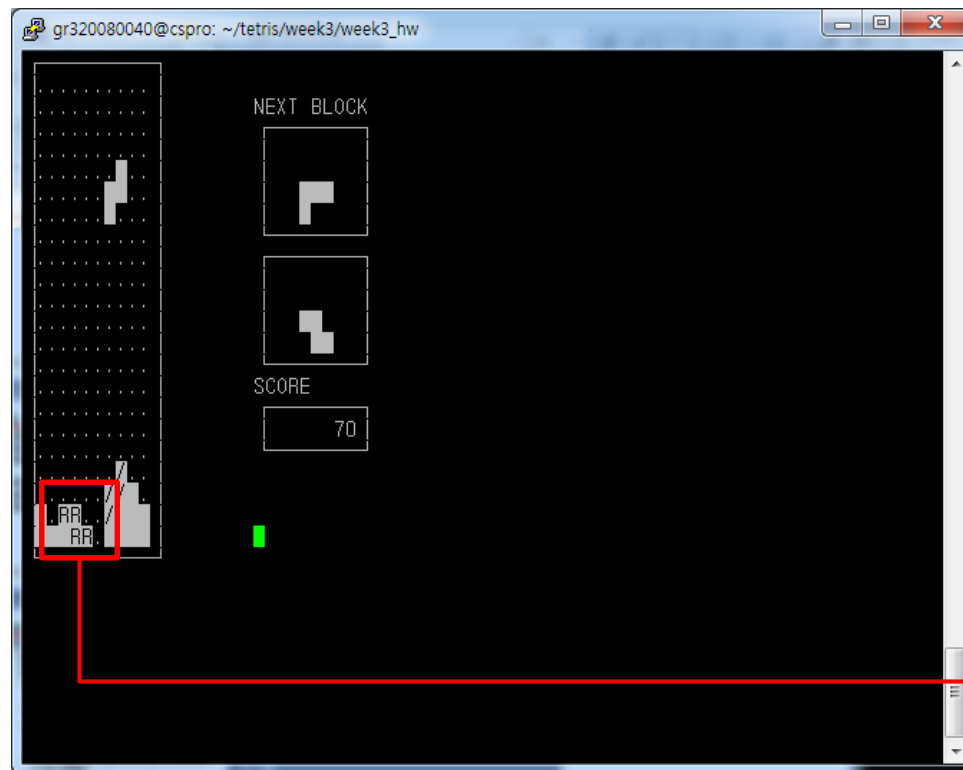
---

## □ 추천 시스템(Recommendation System )

- 테트리스 게임 중, 현재 블록 정보와 블록 미리 보기 기능을 통해 제공되는 next 블록 정보를 사용해서(미래를 고려해서), 사용자가 더 많은 점수를 얻기 위해 현재 필드의 어느 위치에 블록을 놓아야 하는지 추천해주는 추천 시스템을 구현한다.
  - 현재 블록의 정보와 블록 미리 보기 기능의 첫 번째, 두 번째 next 블록의 정보를 이용한다.
  - 블록의 ID, 블록의 회전 수, 블록이 놓여질 위치, 현재의 필드 상태를 고려한다.
  - 추천되는 블록의 위치를 계산하는 과정은 블록이 필드에 놓여지고, 첫 번째 next 블록이 현재 블록으로 바뀔 때 수행된다.
  - 추천되는 블록의 위치는 1주차 숙제로 구현된 그림자 기능을 이용해서 필드 상에 보여준다(그림자 기능은 그대로 유지하고, 추가적으로 구현)

# 테트리스 프로젝트 3주차 구현결과(1/2)

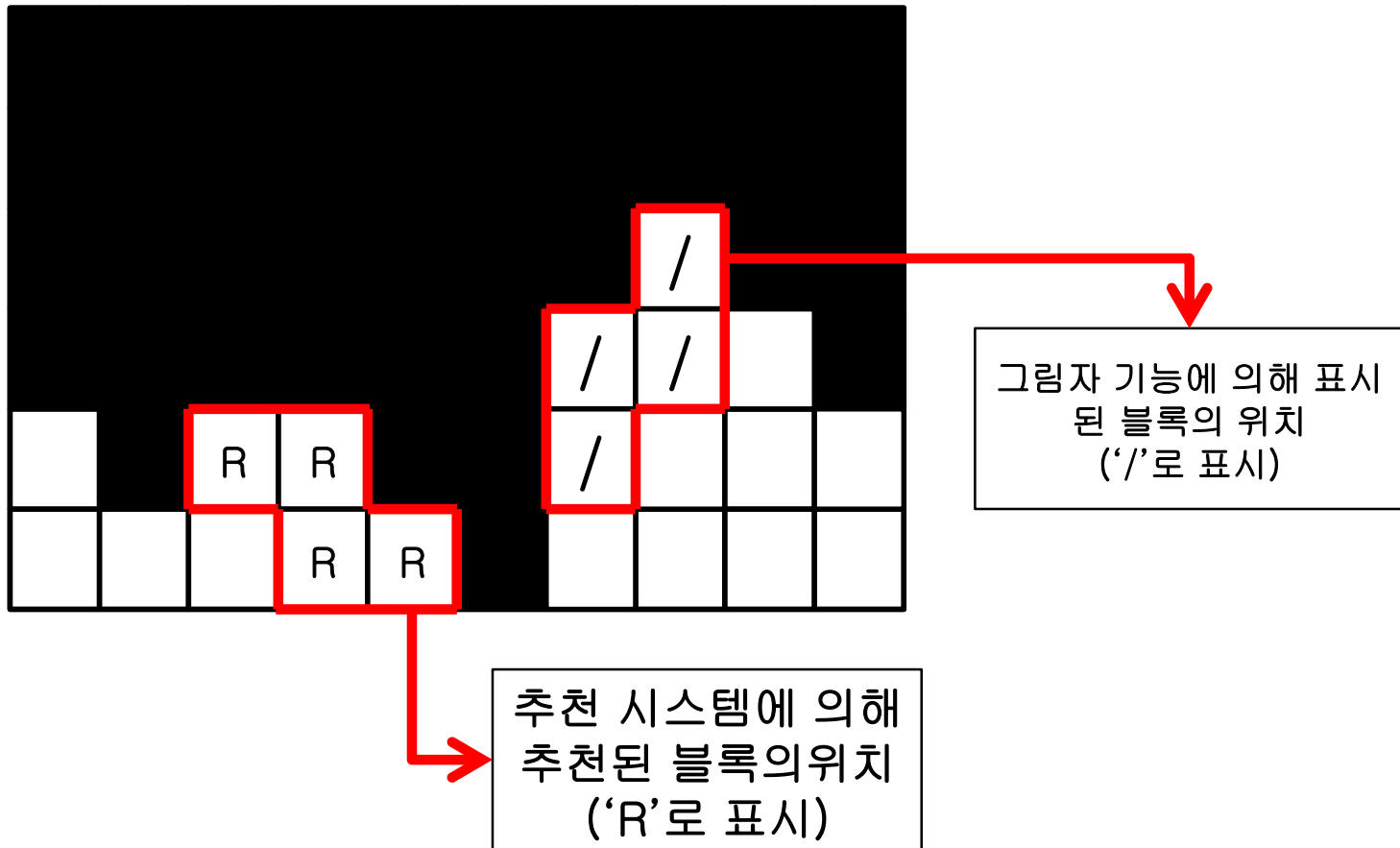
- 추천 시스템 - 사용자에게 좋은 점수를 받을 수 있는 블록의 위치와 회전수를 추천한다.
  - 블록이 추천되는 위치는 character 'R'을 이용해서 나타낸다.



추천된 블록  
그림자 기능('/')과 구분  
- 'R'로 블록 채우기

## 테트리스 프로젝트 3주차 구현결과(2/2)

- 앞의 그림을 실행화면을 확대하면 다음과 같다.



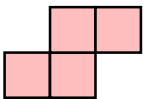
---

# 테트리스 프로젝트 3주차

## 구현 프로그램 설명, Flow Chart 및 함수표

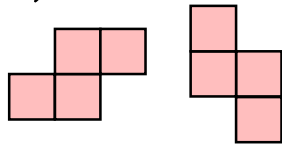
Function() : 구현할 함수

# 추천 시스템 예제(1/7)

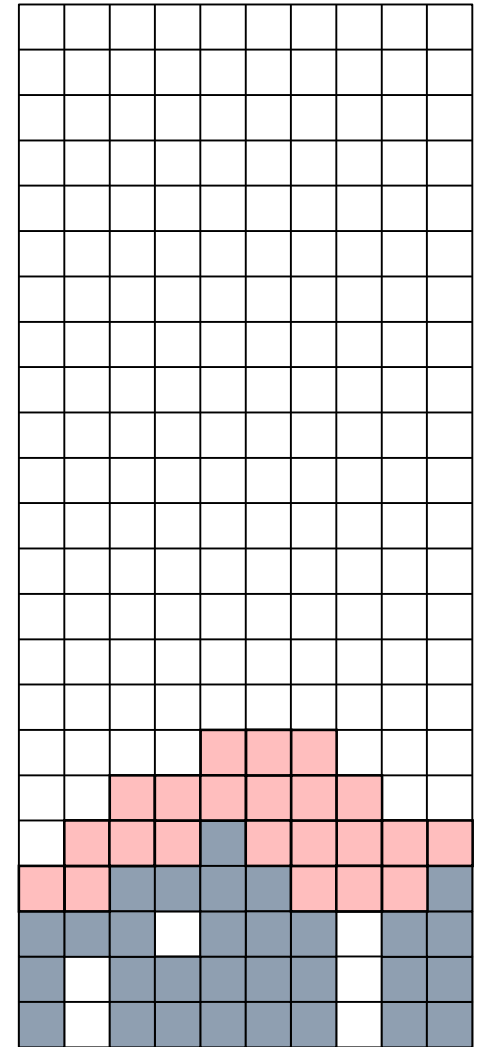
□ 현재 블록 :  회전수: 0

□ 현재 블록을 놓을 수 있는 필드상의 위치가 8개.

□ 회전 수는 0, 1이므로 총 2가지 경우가 존재한다.

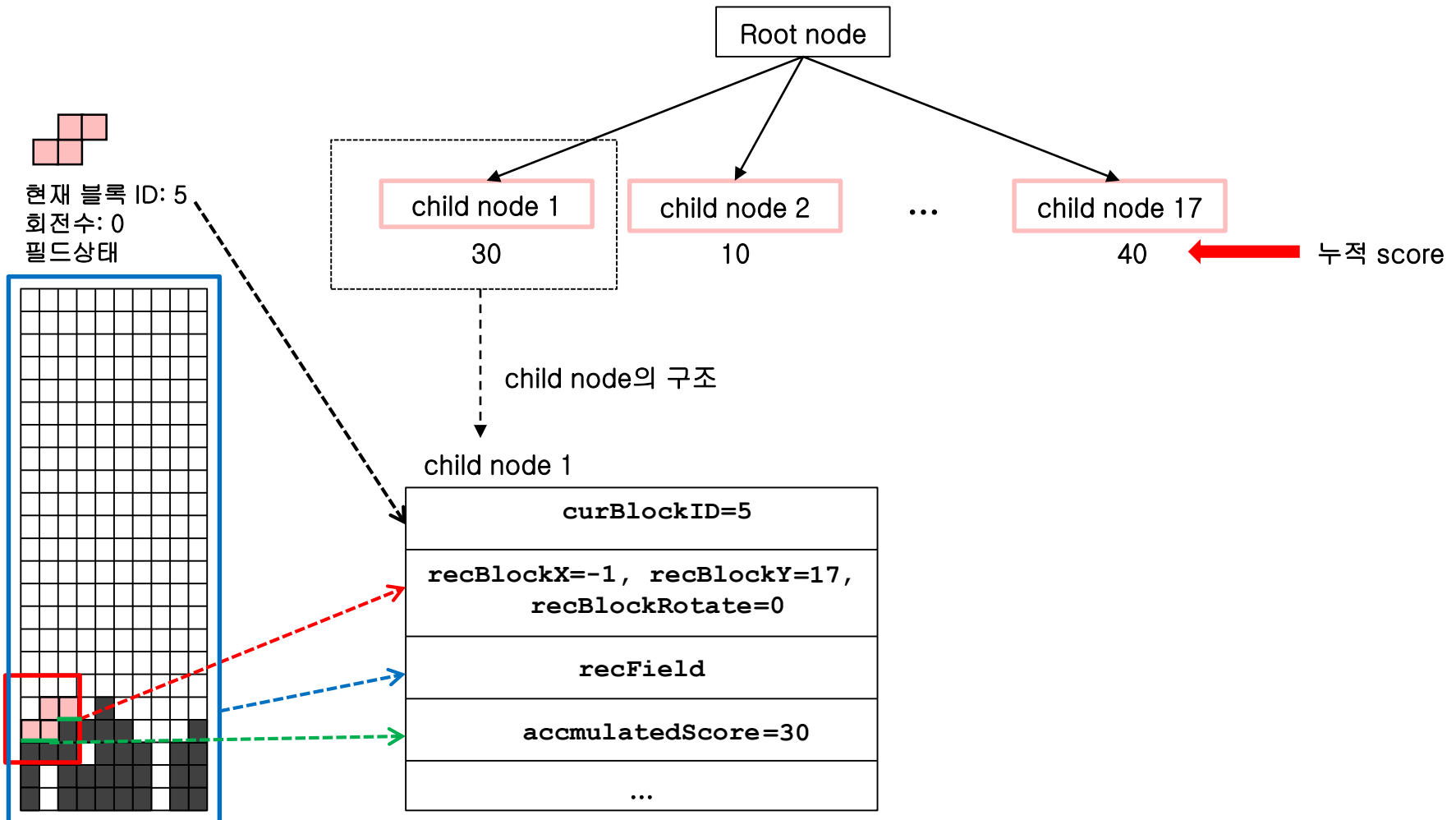


□ 각 회전에 대해, 필드상에 놓을 수 있는 위치는, 8, 9이므로 총 17가지 경우가 존재한다.




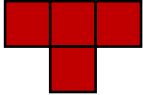
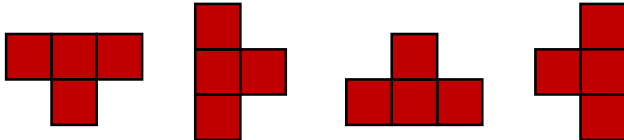
# 추천 시스템 예제(2/7)

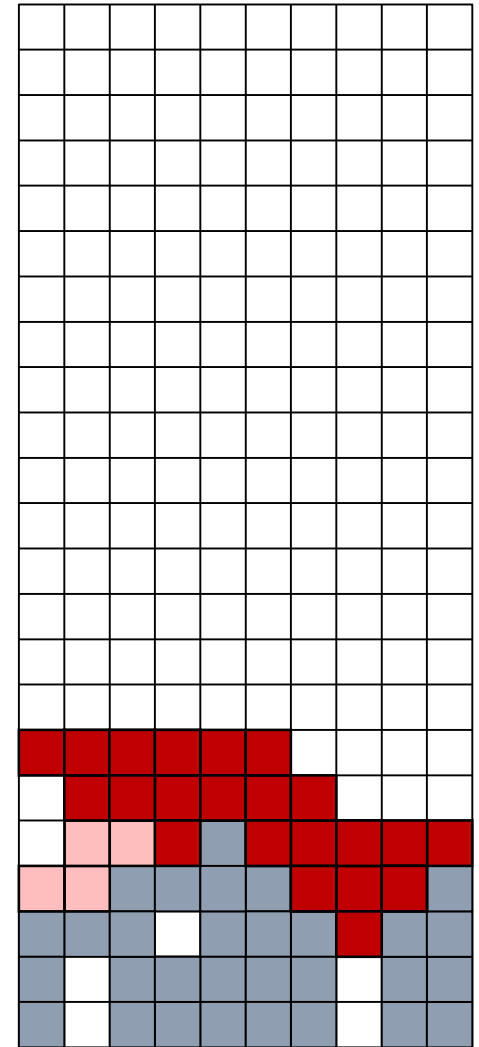
- 고려되는 17개의 경우를 tree 구조로 구성하는 과정과 node의 구조.





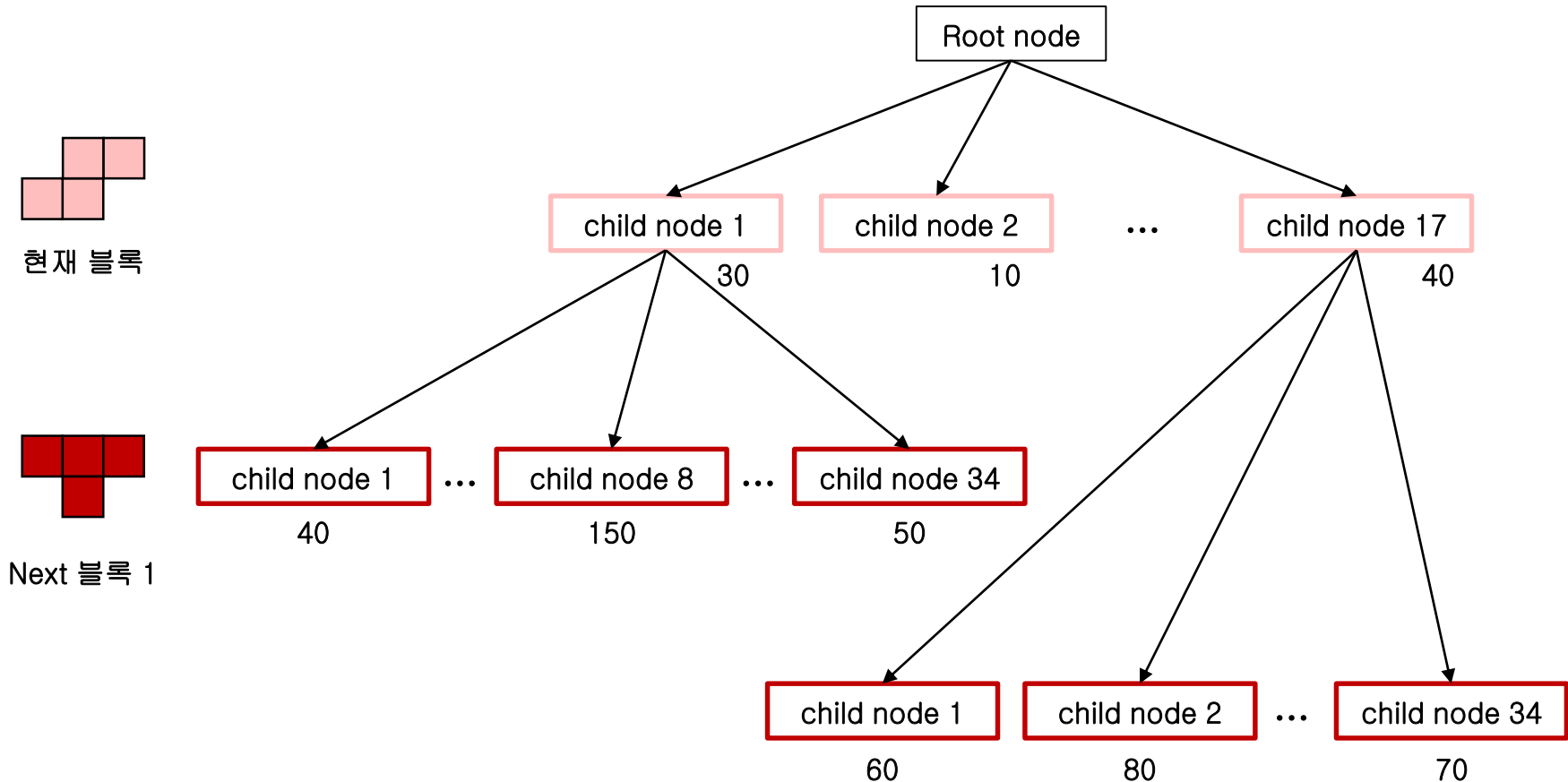
## 추천 시스템 예제(3/7)

- 가정 : 현재 블록  이 놓여진 상황 가정
- Next 블록 1 :  회전수: 0
- 현재 블록을 놓을 수 있는 필드상의 위치가 8개.
- 회전 수는 0, 1, 2, 3이므로 총 4가지 경우가 존재한다. 
- 각 회전에 대해, 필드상에 놓을 수 있는 위치는, 8, 9, 8, 9이므로 총 34가지 경우가 존재한다.



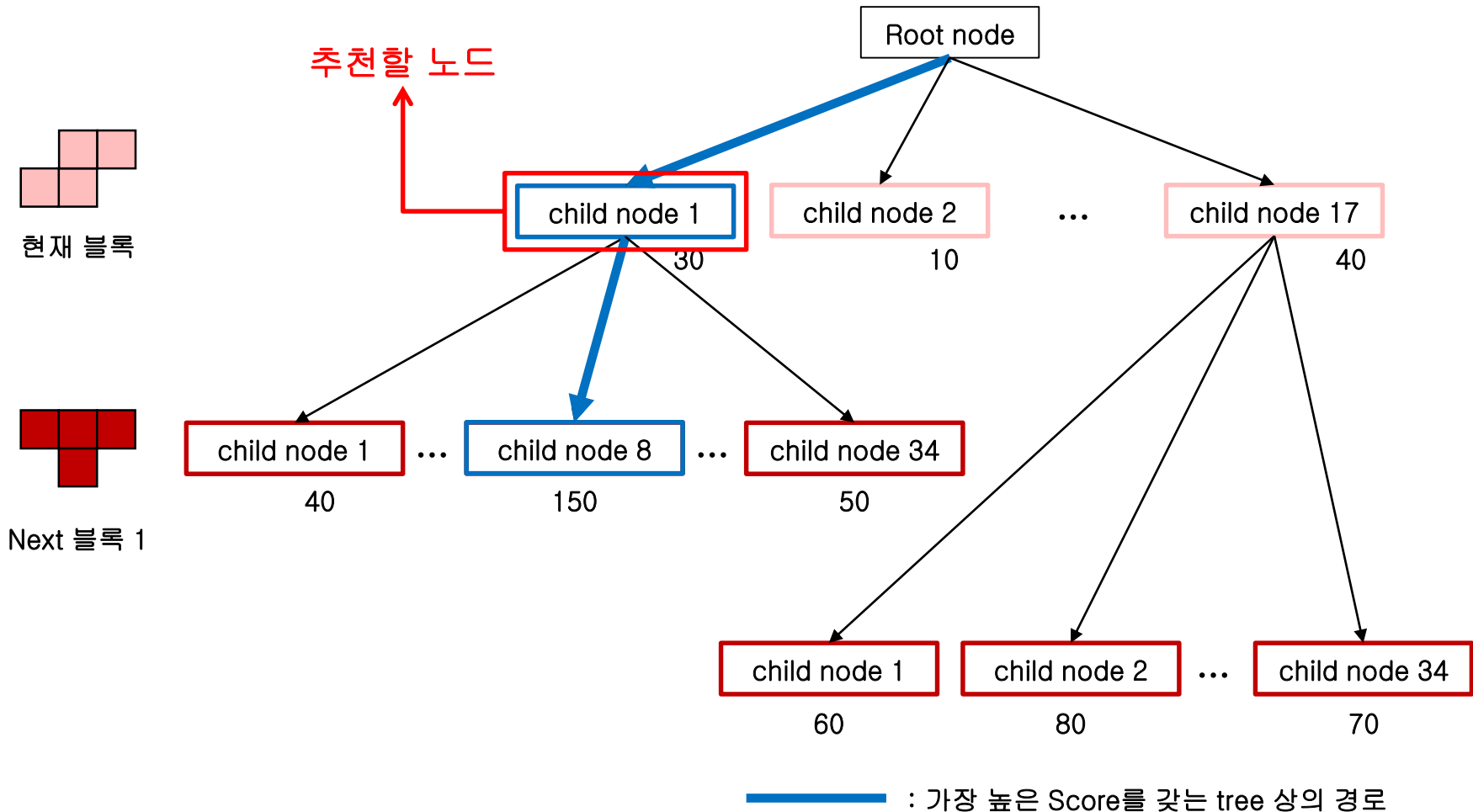
# 추천 시스템 예제(4/7)

- 고려되는 34개의 경우를 tree 구조로 구성하는 과정



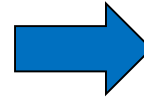
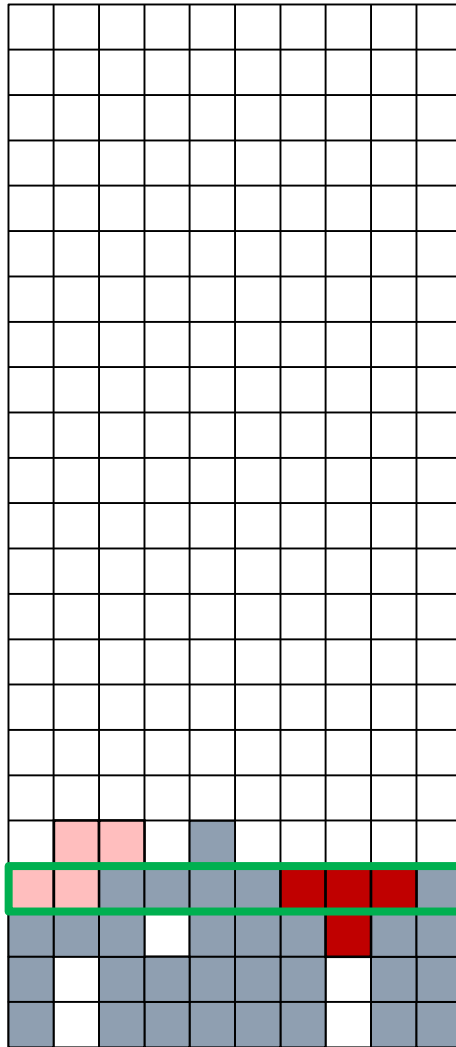
# 추천 시스템 예제(5/7)

□ 최고의 점수를 달성하는 경로



# 추천 시스템 예제(6/7)

가장 높은 누적  
score를 갖는  
tree 상의 경로를  
따라 블록을 놓은  
경우



Accumulated score

= (첫 번째 블록이 필드와 맞닿은 면적) \* 10

+ (두 번째 블록이 필드와 맞닿은 면적) \* 10

+ (삭제된 라인수)<sup>2</sup> \* 100

= 3 \* 10

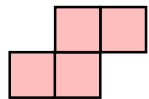
+ 2 \* 10

+ 1<sup>2</sup> \* 100

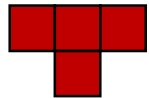
= 150

→ 한 줄 삭제됨

# 추천 시스템 예제(7/7)



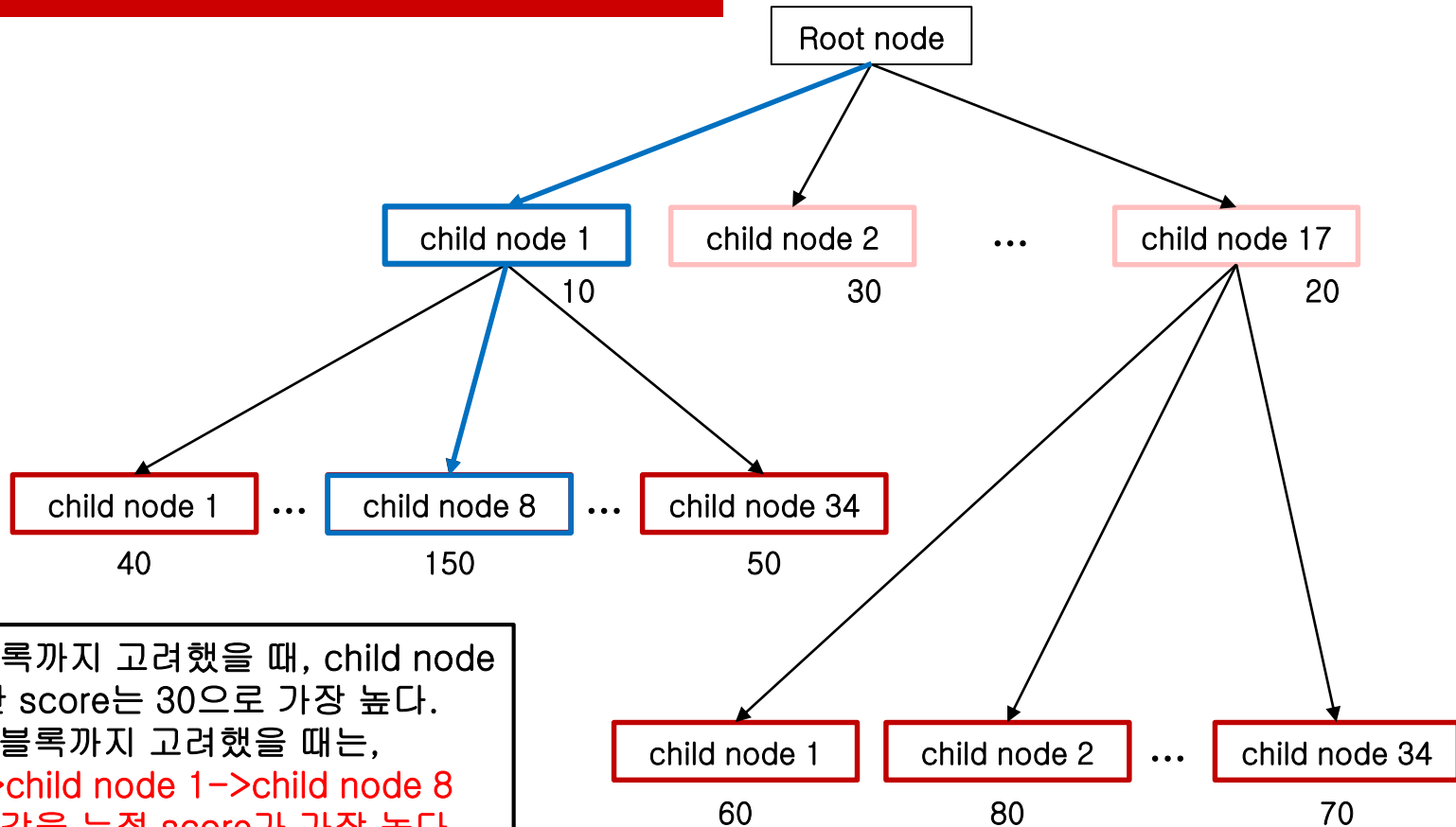
현재 블록



Next 블록 1

비록 현재 블록까지 고려했을 때, child node 2에서 달성한 score는 30으로 가장 높다. 하지만 next 블록까지 고려했을 때는, root node → child node 1 → child node 8 경로를 따라 갔을 누적 score가 가장 높다.

따라서 미래를 알 수 있거나 예측할 수 있다면, 현재 상황에서 더 나은 결정을 내릴 수 있다.



## 추천 시스템의 tree 구조의 비효율성 예제(1/5) – 계산속도

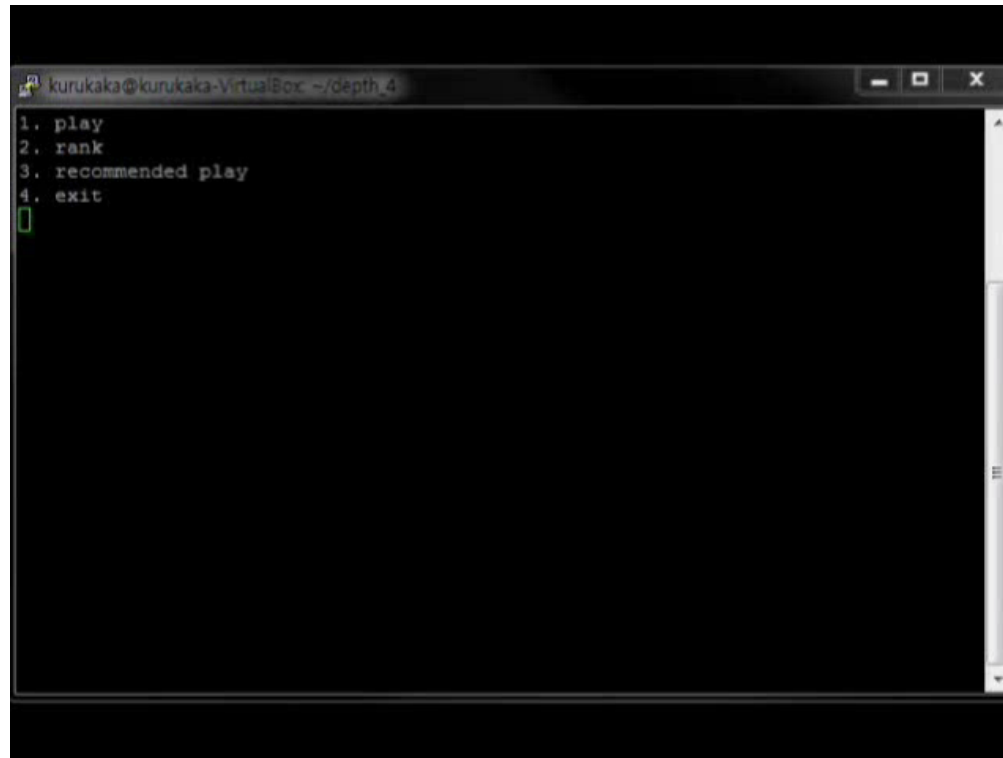
- 현재 블록과 미리 보기로 볼 수 있는 2개의 블록을 고려했을 때(tree의 depth가 3일 때)



```
kurukaka@kurukaka-VirtualBox ~/depth_3
1. play
2. rank
3. recommended play
4. exit
```

## 추천 시스템의 tree 구조의 비효율성 예제(2/5) – 계산속도

- 현재 블록과 미리 보기로 볼 수 있는 3개의 블록을 고려했을 때(tree의 depth가 4일 때)
  - Depth가 3인 경우보다 계산량이 많아서 속도가 느리다.



```
kurukaka@kurukaka-VirtualBox ~/depth_4
1. play
2. rank
3. recommended play
4. exit
█
```

## 추천 시스템의 tree 구조의 비효율성 예제(3/5) – score

- 현재 블록과 미리 보기로 볼 수 있는 2개의 블록을 고려했을 때(tree의 depth가 3일 때)





## 추천 시스템의 tree 구조의 비효율성 예제(4/5) – score

- 현재 블록과 미리 보기로 볼 수 있는 3개의 블록을 고려했을 때(tree의 depth가 4일 때)
  - Depth가 3일 때보다 더 큰 score를 달성한다.



# 추천 시스템의 tree 구조의 비효율성 예제(5/5)

- 위의 예제와 영상에서는 가능한 모든 경우(play 시퀀스, 블록과 블록의 위치의 나열)를 표현하는 tree를 구성한다.
  - Play 시퀀스: (블록1, 블록1 위치), (블록2, 블록2 위치), ...
- Depth가 3인 경우와 Depth가 4인 경우의 비교
  - Depth가 3일 때, depth가 4를 고려할 때보다 블록의 위치 추천을 위한 계산의 속도는 더 빠르다.
  - Depth가 4일 때, depth가 3일 때보다 더 나은 score를 달성한다.
    - Score 비교(game over시)
      - Tree의 depth가 2일 때, 5920
      - Tree의 depth가 3일 때, 17660
      - Tree의 depth가 4일 때, 42780
      - Tree의 depth가 5일 때, 현저히 속도가 느려짐.
- 장, 단점
  - 장점: 추천시스템에서 더 많은 블록을 고려하게 되면, 더 나은 score를 달성할 수 있다.
  - 단점: 추천 블록의 위치를 계산하기 위한 시간 및 공간의 소모가 심각하다.

# 추천 시스템의 tree 구조의 비효율성(1/2)

## □ 시간

- 추천 시스템을 구축을 위해 2번째 next 블록까지 고려해서 tree를 구성한다고 가정하자.
- 각 블록에 대해서 최대 34가지 블록의 위치가 고려된다고 가정하면, tree의 leaf 노드의 수는  $34^3=39304$ 개이다.
- 만약 블록을 총 4개 고려하게 되면,  $34^4=1336336$ 개이다.
- 값 39304는 39304개의 다른 경로(path)가 존재한다는 것을 의미한다.
- 테트리스 게임에서 더 좋은 점수를 얻기 위해서는 아주 먼 미래를 고려해야 하는데, 이와 같이 지수적(exponential)으로 늘어나는 경로의 수( $34^n$ ,  $n$ 개의 고려하는 블록의 수)를 찾으면서 자료구조를 구축하는 것은 너무 많은 시간과 노력이 필요하기 때문에 처리하기 어렵다.

- ## □ 따라서 이와 같은 노력을 줄이면서 좋은 점수를 낼 수 있는 새로운 방법이 필요하다.

## 추천 시스템의 한 tree 구조의 비효율성(2/2)

### □ 공간

- 추천 시스템의 tree의 각 노드는 많은 정보를 포함하고 있다.
- 각 노드의 정보인 블록의 ID, 블록의 위치, 블록의 회전 수, 필드 정보, 누적 점수 등을 저장하기 위해 많은 메모리 공간을 할애하고 있다.
- 하나의 노드에 필요한 메모리 공간이  $c$ 라고 가정하면, 3개의 블록이 고려될 때, 사용되는 메모리 공간은  $c \times 39304$ 이고, 4개의 블록이 고려될 때, 사용되는 메모리 공간은  $c \times 1336336$ 이다.
- 만약  $n$  개의 블록을 고려한다면,  $c \times 34^n$  의 메모리 공간을 사용해야 한다. 이와 같은 많은 공간의 사용은 메모리 overflow 등 다른 문제를 야기할 가능성이 많다.

- ### □ 따라서 시간의 경우와 마찬가지로 이와 같은 메모리 공간을 줄이면서 좋은 점수를 낼 수 있는 새로운 방법이 필요하다.

# 비효율성을 해결하기 위한 방법 – Pruning(1/4)

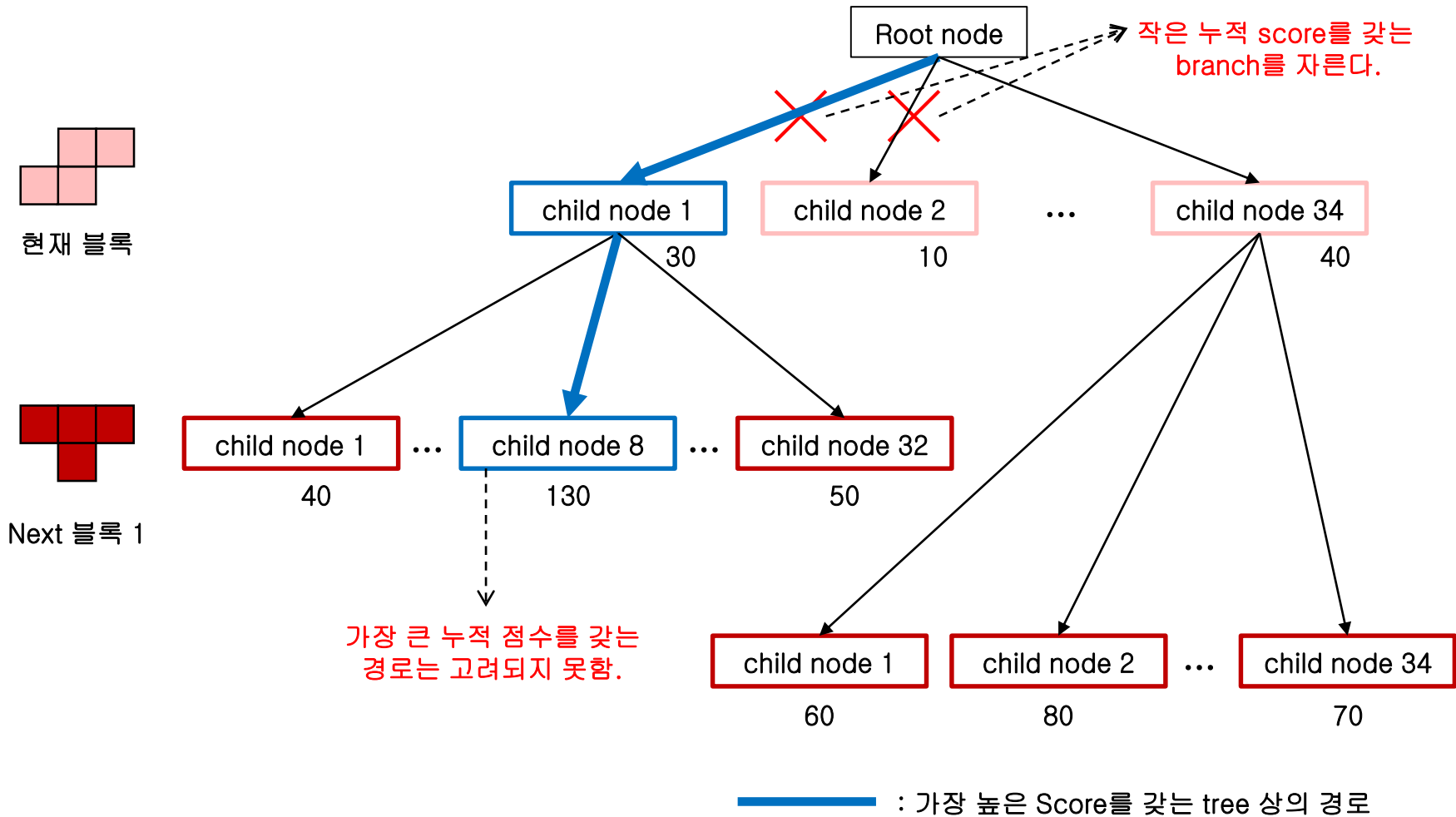
---

## □ Pruning tree

- Tree의 branch를 자른다(prune).
- 장점: 고려되는 node의 감소로 시간 단축되고, 사용되는 공간도 줄어든다.
- 단점: 자른 branch에서 누적 score가 가장 큰 경로가 만들어질 가능성이 있다.
- 단점을 최소화할 수 있는 pruning 방법이 필요하다.

# 비효율성을 해결하기 위한 방법 - Pruning(2/4)

- 단점의 예) tree를 구성할 때마다 작은 score를 갖는 node를 제거



□ 공간

- 
- 4 5 5 4 5 4 3 0 3 4

## □ 공간의 복원

- 이와 같은 복원의 첫점에도 불구하고, 이와 같이 필드를 작은 data를 저장, 복원해서 블록이 놓을 위치를 고려할 수 있다면, 기존에 사용되었던 공간 중 많은 부분을 줄일 수 있다.

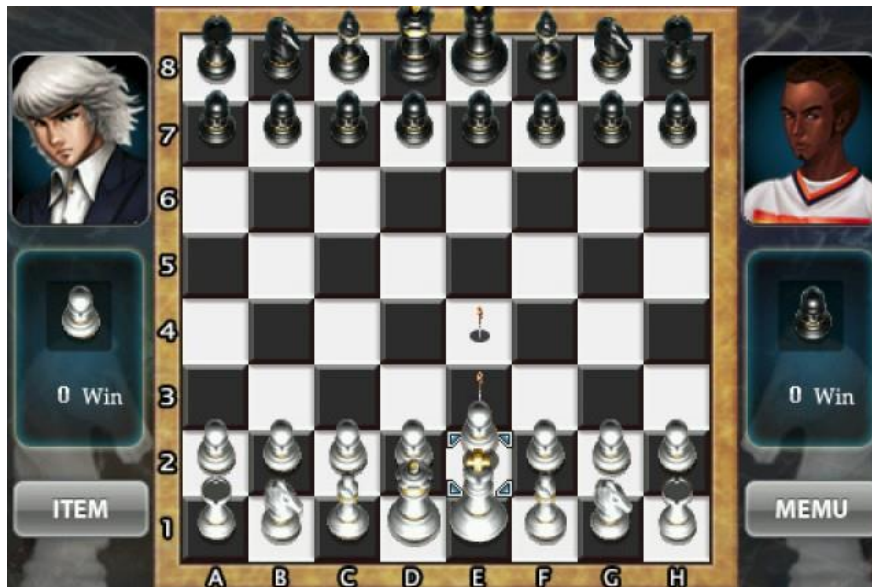




# Tree를 이용한 예측 기법을 사용하는 applications

□ 주위에 친숙한 많은 게임들에서 이와 같은 예측 기법을 사용한다.

- 바둑
- 장기
- 체스
- 오목



# 테트리스 프로그램 전체 흐름과의 관계

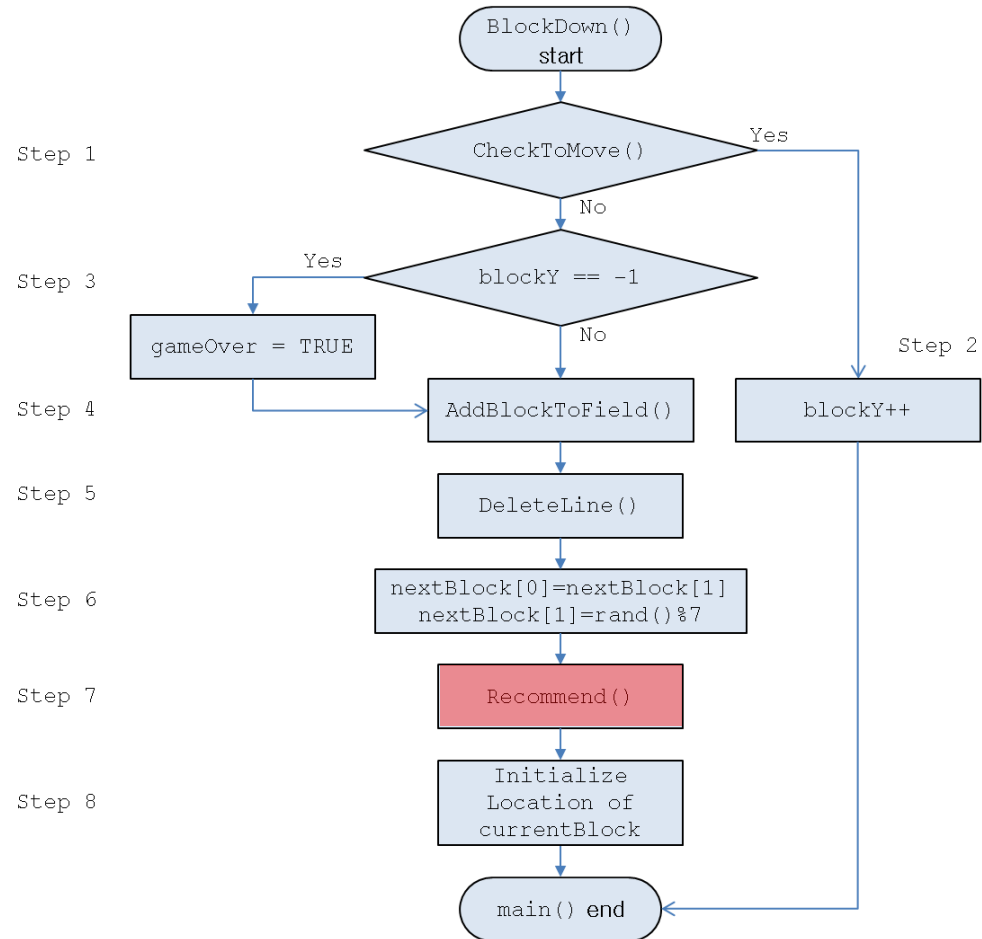
## □ 키 입력에 대한 동작 (BlockDown() 함수)

### ● recommend()

- 사용자에게 좋은 점수를 얻을 수 있는 블록의 위치를 계산하는 함수. 현재와 다음 블록의 모양, 회전 수, 필드의 상태, 얻을 수 있는 점수 등을 고려한다. 이 모든 사항들은 tree를 사용해서 표현한다.

### ● DrawRecommended()

- Recommend() 함수에서 추천된 위치에 블록을 그려주는 함수이다. 이 함수는 DrawBlockWithFeatures() 함수에서 호출된다.



# 추천 시스템 구현을 위한 tree node structure(1/2)

---

```
□ typedef struct _Node {
    //must-have elements
    int accumulatedScore;
    char recField[HEIGHT][WIDTH];
    struct _Node **child;
    //optional element
    int curBlockID;
    int recBlockX, recBlockY, recBlockRotate;
    struct _Node *parent;
    ...
} Node;
```

# 추천 시스템 구현을 위한 tree node structure(2/2)

## □ 추천 시스템 구현을 위한 tree node structure

### ● Must-have element

- `int level` : tree의 level(or depth)를 나타낸다.
- `int accumulatedScore`: 누적된 점수
- `int recField[HEIGHT][WIDTH]`: 추천된 블록의 위치와 회전 수를 고려해서 블록을 테트리스 필드에 놓았을 때의 필드 상태
- `struct _Node **child`: tree에서 children을 가리키는 node pointer 이고, child 수만큼 동적 할당을 한다.

### ● Optional element

- `int curBlockID`: tree에서 고려되는 block의 ID
- `int recBlockX, recBlockY, recBlockRotate`: 추천된 위치와 회전 수
- `struct _Node *parent`: tree에서 부모를 가리키는 노드 포인터 (node pointer)

# 테트리스 프로젝트 3주차 구현내용(1/6)

- 3주차 추천 시스템 구현에서 사용되는 자료구조는 tree이다.
- 구현할 함수: `recommend(Node *root)`
  - 현재블록과 다음 2개의 블록을 고려해서 모든 play 시퀀스를 나타낼 수 있는 tree를 구성하고, tree의 정보를 바탕으로 사용자가 좋은 score를 얻을 수 있는 현재 블록의 위치를 계산하는 기능을 한다.
  - Tree의 구성과 블록의 추천 위치를 찾는 과정은 함께 이루어진다.
  - Tree에서 고려하는 블록의 수를 조정하는 것에 의존하지 않고 함수가 수행되어야 한다.
    - 예를 들어, 블록의 수가 3개(tetris.h에서 `VISIBLE_BLOCKS`)를 고려해서 작성한 `recommend()` 함수는 블록의 수, `VISIBLE_BLOCKS`가 다른 수로 정의되어도 `recommend()` 함수에는 아무런 수정 없이 정상적으로 동작해야 한다.
  - Parameters
    - `Node *root`
      - 현재 고려하는 블록에 대한 child 노드들의 부모(parent) 노드 주소를 갖는 포인터(node pointer). 이는 현재 블록이 놓여질 위치를 결정할 필드 정보와 이전까지 누적된 점수에 접근할 수 있도록 한다.

# 테트리스 프로젝트 3주차 구현내용(2/6)

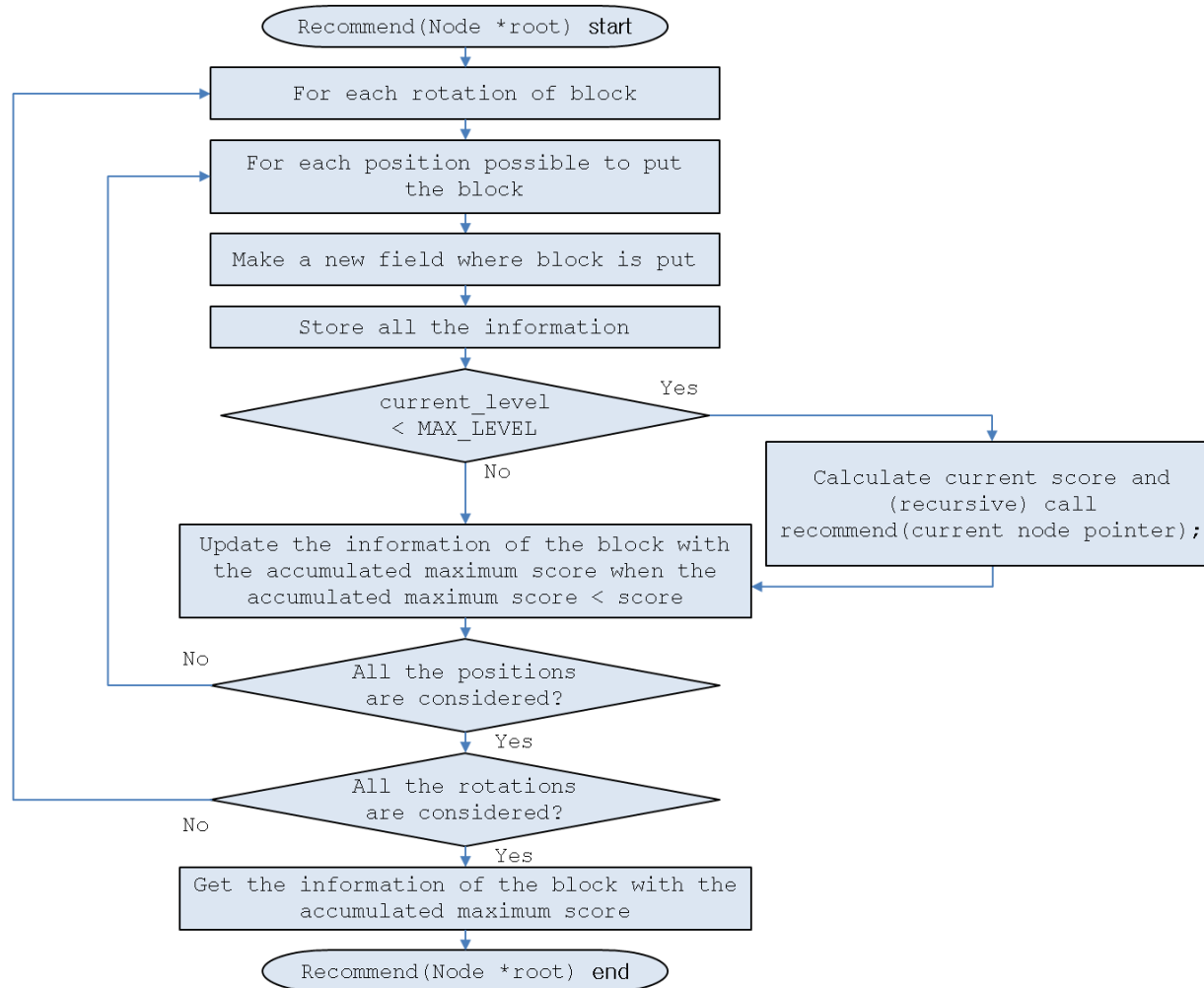
## □ recommend (Node \*root)

- 앞의 예제와 같이 블록, 블록의 회전 수, 필드의 상태, 누적 score를 갖는 node 로 구성된 모든 play 시퀀스를 표현하기 위한 tree를 만든다.
- 구현
  - 각 블록의 회전 수에 대해서(1<sup>st</sup> loop문) 필드상에 놓일 수 있는 위치(2<sup>nd</sup> loop문)를 차례로 고려한다.
    - 노드를 구성하는 정보를 저장한다.
    - 현재 고려하는 level이 최대 level수(최대 고려블록 수)보다 작으면, 누적 score를 계산하기 위해서 재귀적 함수 호출을 한다.

Call Recommend(current node address) ;
    - 현재 고려하는 level이 최대 level수(최대 고려블록 수)와 같으면, accumulatedScore에 현재 점수를 저장한다.
    - 현재까지 최대 누적 점수와 현재 얻어진 점수를 비교해서, 가장 큰 누적 점수를 갖는 블록의 회전수와 위치를 갱신한다.
  - 2<sup>nd</sup> loop 종료
  - 1<sup>st</sup> loop 종료
  - 누적 score가 가장 큰 경로 상에 존재하는 현재 블록의 추천된 위치와 회전수를 기억하여 블록의 위치를 추천한다.

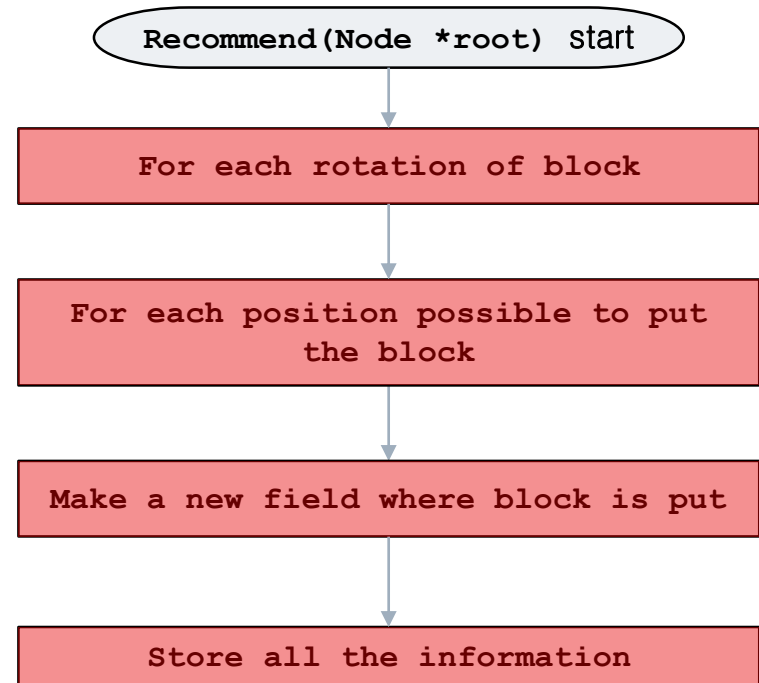
# 테트리스 프로젝트 3주차 구현내용(3/6)

## □ recommend() flow chart



# 테트리스 프로젝트 3주차 구현내용(4/6)

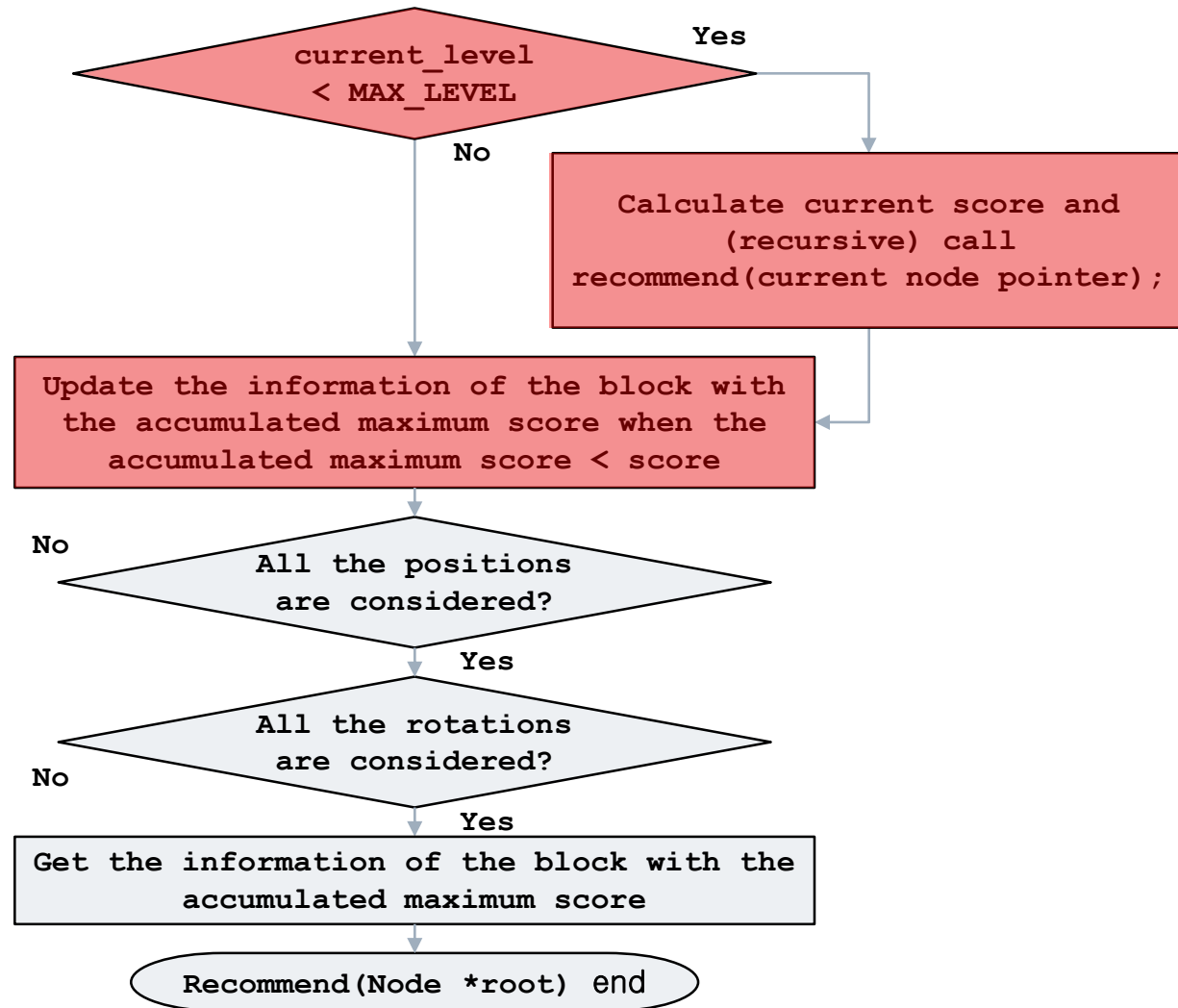
- 현재 고려하는 블록의 회전수에 대한 loop 문을 시작한다.
- 각 블록이 놓일 수 있는 위치를 세고, 위치에 대한 loop문을 시작한다.
- 주어진 회전수를 갖는 블록을 해당 위치에 놓았을 때, 필드가 어떻게 바뀌는지 처리 한다.
- 모든 정보를 노드에 저장한다.





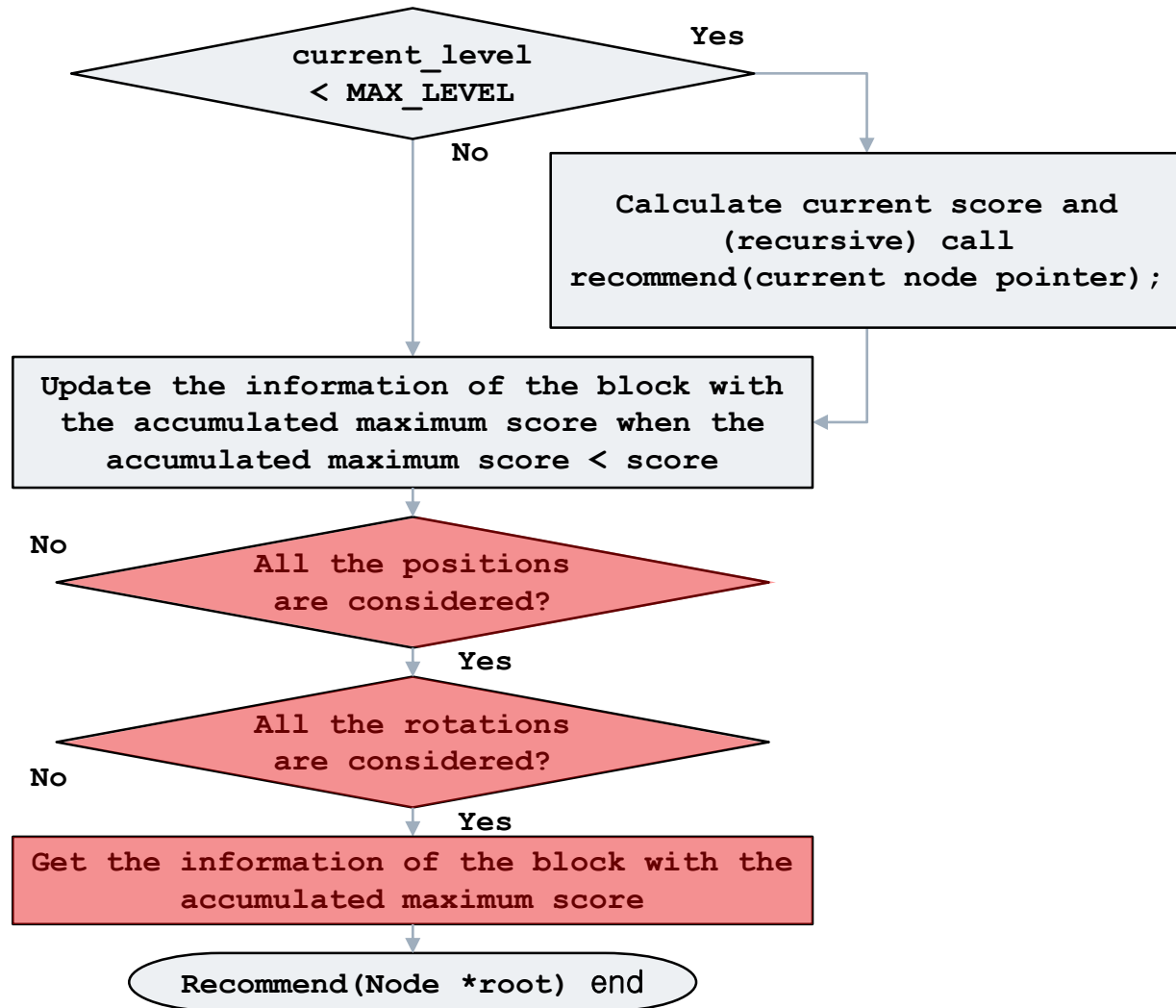
# 테트리스 프로젝트 3주차 구현내용(5/6)

- 블록을 모두 고려하면(tree의 최대 level에 도달하면), 현재 점수를 계산하고, 그렇지 않으면 누적 점수 계산을 위해 재귀적 함수 호출을 한다.
- 이 과정이 끝나면, 지금까지 얻어진 점수와 현재 고려하는 노드의 누적 점수를 비교해서 업데이트 하고, 추천할 블록의 위치와 회전수를 기억한다.



# 테트리스 프로젝트 3주차 구현내용(6/6)

- 모든 블록의 위치가 고려되었는지 체크한다
- 모든 회전수가 고려되었는지 체크한다.
- 최대 누적 점수를 갖는 블록의 위치와 회전수를 얻는다.



# 테트리스 프로젝트 3주차 실습 평가

---

## □ 함수 `recommend()`

- 모든 play 시퀀스가 고려되어 tree가 구성되는가?
- 구성된 tree에서 누적 score가 가장 높은 경로를 찾고, 현재 블록이 놓여질 위치에 대한 정보를 정확히 추출해내는가?
- 화면에 정확하게 그 위치를 표시하는가?
- `VISIBLE_BLOCKS`의 값이 수정되어도 `recommend()` 함수는 아무런 수정 없이 잘 동작하는가?

## □ 함수 `modified_recommend()`

- 새로운 tree를 구성하여 테트리스 play한 경우 일정  $t$  시간 동안 얻은 누적 score를  $\text{score}(t)$ 라 하고,  $t$ 시간 동안 tree의 정보를 구성하기 위해 소비된 누적 시간  $\text{time}(t)$ 이라고 하면,
- 효율성 =  $\text{score}(t)/\text{time}(t)$
- 효율성이 큰 tree를 구현한 학생에게 큰 점수를 부여한다.

# 테트리스 프로젝트 3주차 숙제1

---

- `recommend()` 함수를 구현하고 난 후에는 앞에서 설명한 pruning, 데이터의 단순화 등 자신만의 방법으로 효율적인 tree를 새롭게 구성한다.
- 새롭게 자신만의 구성된 tree를 통한 추천 시스템은 `modified_recommend()` 라는 함수 이름으로 작성한다.

# 테트리스 프로젝트 3주차 숙제2

---

- 메뉴의 3번의 recommended play모드를 구현한다.
  
- recommended play 모드: 이 모드는 테트리스 게임 play를 추천 기능을 통해 계산된 위치에 블록을 놓는 자동화된 play 모드이다.
  - 사용자는 조작할 수 없다.
  - 추천기능으로 계산된 필드 상의 위치에 현재 블록을 놓는다.
  - 이 모든 과정을 화면으로 인지할 수 있는 속도로 보여준다.
  
- 구현
  - Flag MENU\_REC\_PLAY를 사용해서 메뉴 3번 기능 동작하도록 하고, 3번을 선택했을 때, 이 모드를 수행할 수 있도록 한다.
  - 이 모드는 기존의 play()에서 수행하는 flow와 매우 비슷하게 구현한다.
  - 수행할 수 있는 commend는 오직 종료(Quit)이다.
  - 블록이 추천되는 위치를 계산하고, 블록이 놓여질 위치에 놓는 동작을 반복한다.

# 테트리스 프로젝트 3주차 숙제 flow chart

