실전 개발형 코딩 테스트 문제 풀이 ②

문제 풀이를 위한 설계

문제 풀이를 위한 설계 | 문제 해결 방법 알아보기

강사 나동빈

실전 개발형
코딩 테스트 문제 풀이 ②

문제 풀이를 위한 설계

개발형 코딩 테스트
문제 풀이를 위한 설계

틱택토(Tic Tac Toe)

- 3 × 3 바둑판에서 진행하는 보드 게임으로, 간단히 **3목**으로 생각할 수 있다.
- 하나의 **행**, 하나의 **열**, 혹은 하나의 **대각선**을 차지하는 경우 즉시 승리한다.

O	X	O
X	O	O
X	O	X

무승부

O	X	X
O	O	O
X	O	X

플레이어 O 승리

O	O	X
O	X	
X	O	X

플레이어 X 승리

O	X	O
X	O	
X		O

플레이어 O 승리

O	X	O
O	X	
	X	

플레이어 X 승리

X	X	O
O	O	X
X	O	O

무승부

개발형 코딩 테스트
문제 풀이를 위한 설계

틱택토(Tic Tac Toe)

- 3×3 바둑판에서 진행하는 보드 게임으로, 간단히 **3목**으로 생각할 수 있다.
- 처음에 돌을 놓을 수 있는 위치는 9가지로, 모든 상태(state)의 수를 고려하면 다음과 같다.
- $9! = 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 362{,}880$
- 이처럼 총 경우의 수가 한정적이므로, 모든 경우의 수를 다 고려하는 것이 가능하다.

최소 최대(Minimax) 알고리즘은 언제 사용할까?

- 두 명의 게임 참가자가 서로 번갈아 가며 다투는 **제로섬(zero-sum) 게임**에서 사용할 수 있는 알고리즘이다.
- 한 명의 승리는 다른 한 쪽의 패배를 의미한다.
- 따라서 플레이어 A에게 유리한 수는 다른 플레이어 B에게 불리한 수가 된다.

개발형 코딩 테스트
문제 풀이를 위한 설계

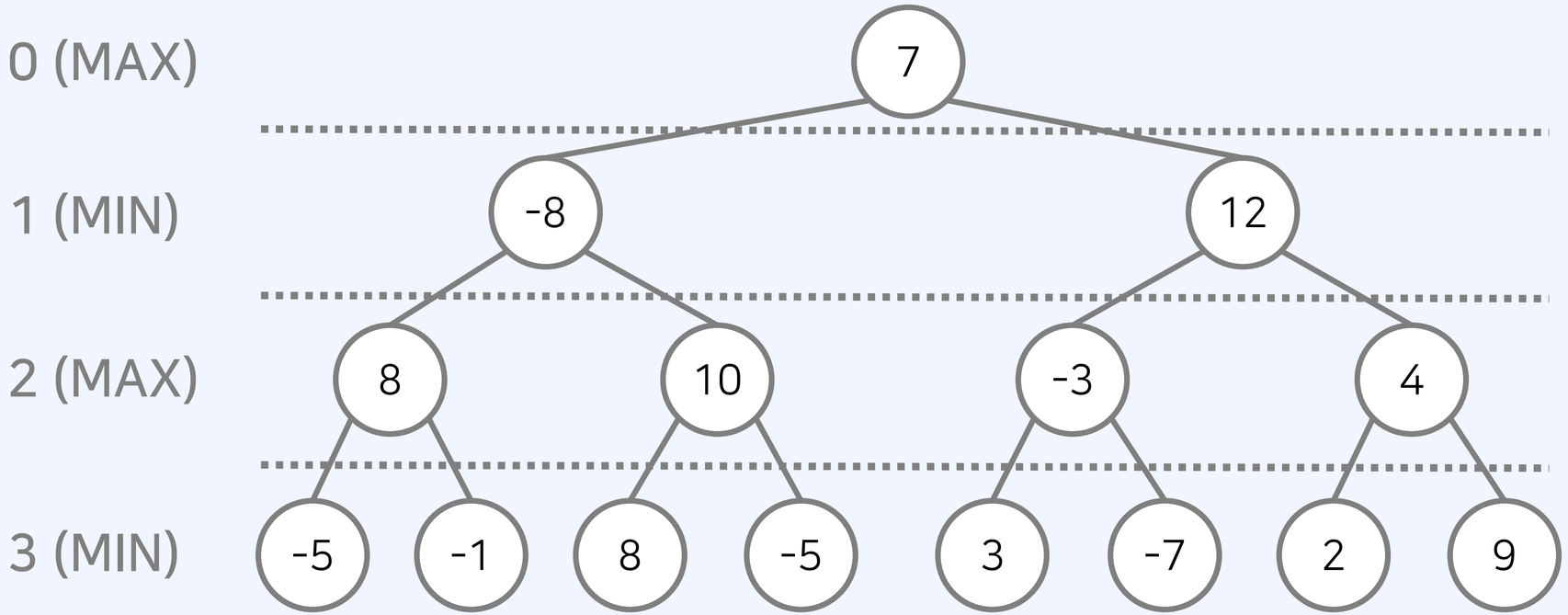
최소 최대(Minimax) 알고리즘 동작 방식

- 플레이어 A에 대한 효용(utility) 함수를 정의할 수 있다.
- 플레이어 A는 효용(utility)를 **최대화(MAX)**하는 수를 선택한다.
- 플레이어 B는 효용(utility)를 **최소화(MIN)**하는 수를 선택한다.

개발형 코딩 테스트
문제 풀이를 위한 설계

최소 최대(Minimax) 알고리즘 동작 방식

- MAX 플레이어는 효용을 최대화, MIN 플레이어는 효용을 최소화한다.
- 최소 최대(Minimax) 알고리즘은 재귀 함수(깊이 우선 탐색)를 이용해 구현할 수 있다.



최소 최대(Minimax) 알고리즘 사용하기

- 체스(chess)와 같이 복잡도가 큰 게임에서는 모든 경우의 수를 탐색하기 어렵다.
- 휴리스틱 접근(어느 정도 깊이의 수까지 탐색한 후에 판정)을 사용할 수 있다.
- 하지만 틱택토(Tic Tac Toe)와 같은 비교적 간단한 게임에서는 **최소 최대 알고리즘**을 이용해 정답을 해결할 수 있다.

개발형 코딩 테스트
문제 풀이를 위한 설계

효용(utility) 함수 정의

- 플레이어 A에 대한 점수(score)를 판정하기 위한 함수로 사용할 수 있다.
- 상황에 따라서 적절한 점수를 반환하도록, 원하는 방식에 따라서 설계할 수 있다.
- **예시)** 플레이어 A가 이기는 경우 +1, 플레이어 A가 지는 경우 -1, 무승부라면 0을 반환

```
function calculate_utility():
    # 플레이어 A가 승리하는 경우
    if win(player_A):
        return 1
    # 플레이어 A가 패배하는 경우
    elif defeat(player_A):
        return -1
    # 아무도 승리하지 않은 경우
    else:
        return 0
```

개발형 코딩 테스트
문제 풀이를 위한 설계

최소 최대(Minimax) 알고리즘의 구현

- 깊이(depth)가 제한된 상황에서의 **최소 최대(minimax) 알고리즘**은 다음과 같다.

```
function minimax(node, depth, maximizing_player) is
    # 미리 설정한 깊이만큼 호출했거나, 게임이 종료된 경우
    if depth == 0 or node is terminal node then
        return the heuristic value of node (utility value)
    if maximizing_player then # MAX 플레이어인 경우
        value = −∞
        # 현재 상황에서 가능한 모든 선택지를 확인하며
        for each child of node do
            # 효용(utility)을 최대화하는 선택지를 고르기
            value = max(value, minimax(child, depth - 1, FALSE)
        return value
    else # MIN 플레이어인 경우
        value = +∞
        # 현재 상황에서 가능한 모든 선택지를 확인하며
        for each child of node do
            # 효용(utility)을 최소화하는 선택지를 고르기
            value = min(value, minimax(child, depth - 1, TRUE)
        return value
```

최소 최대(Minimax) 알고리즘을 사용하는 이유

- 본 문제는 최소 최대 알고리즘을 이용해 해결할 수 있다.
- 결과적으로 **최소 최대(minimax)** 알고리즘은 특정한 플레이어가 현재 상태(status)에서 어떤 선택을 해야 가장 효율적인지 알려준다.
- 이때 각 선택에 따라서 도출되는 효용(utility) 점수를 계산할 수 있다.
- 매 상황에서 MAX 플레이어는 효용을 최대화하는 선택지를 고르게 되고, MIN 플레이어는 효용을 최소화하는 선택지를 고르게 된다.