2020 OSS 개발자 포럼 겨울 캠프

AlphaZero 오목 AI - Day 1

옥찬호

Nexon Korea, Microsoft MVP

utilForever@gmail.com

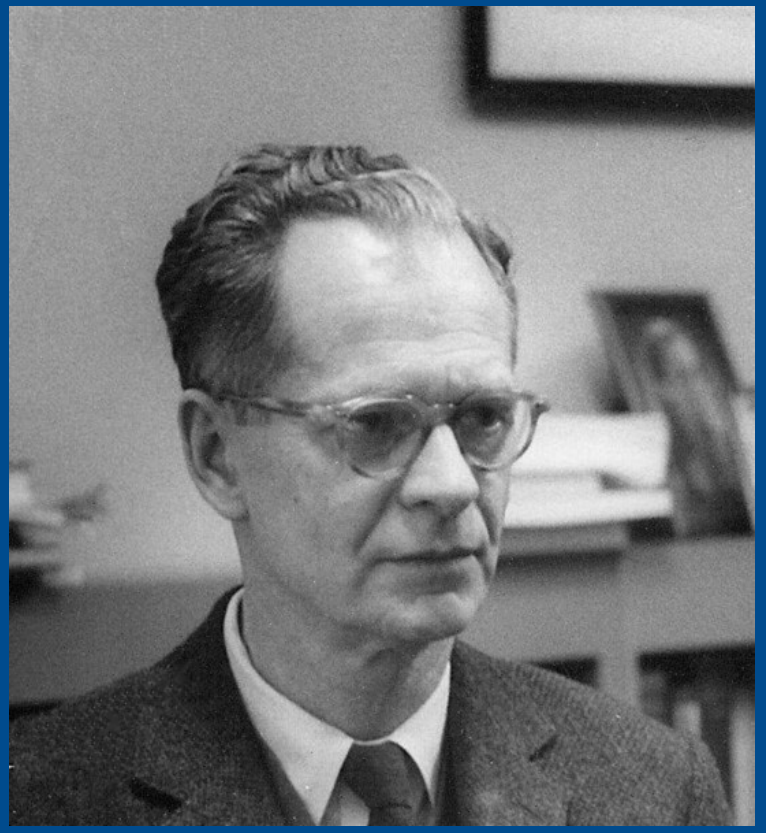
참고

- 오늘 코드는 ‘Deep Learning and the Game of Go’ (Manning, 2019)을 기반으로 변형해서 만들었습니다.
- 발표 준비를 도와준 조교 김현수/박준영 학생에게 감사의 말씀을 드립니다.

오늘 다룰 내용

- 강화학습이란?
- 오목 게임 만들기
- 간단한 오목 AI 봇 만들기
- MCTS(Monte-Carlo Tree Search)

강화학습이란?

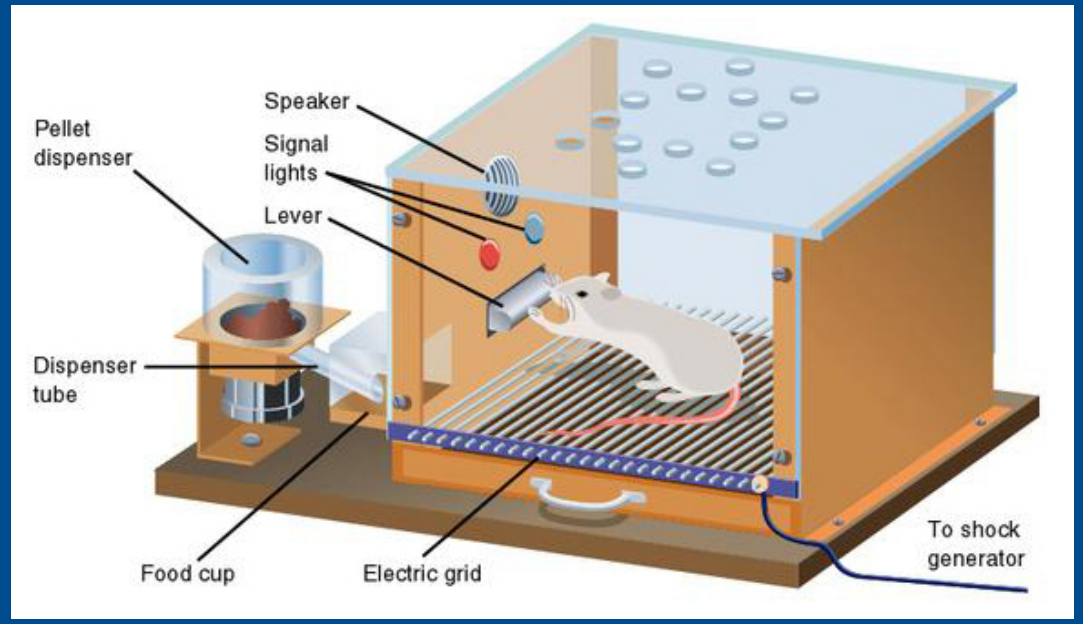


B. F. Skinner (1904~1990)

스키너의 강화 연구

2020 OSS Winter
AlphaZero 오목 AI - Day 1

1. 굶긴 쥐를 상자에 넣는다.
2. 쥐는 돌아다니다가 우연히 상자 안에 있는 지렛대를 누르게 된다.
3. 지렛대를 누르자 먹이가 나온다.
4. 지렛대를 누르는 행동과 먹이와의 상관관계를 모르는 쥐는 다시 돌아다닌다.
5. 그러다가 우연히 쥐가 다시 지렛대를 누르면
   쥐는 이제 먹이와 지렛대 사이의 관계를 알게 되고
   점점 지렛대를 자주 누르게 된다.
6. 이 과정을 반복하면서 쥐는 지렛대를 누르면
   먹이를 먹을 수 있다는 것을 학습한다.



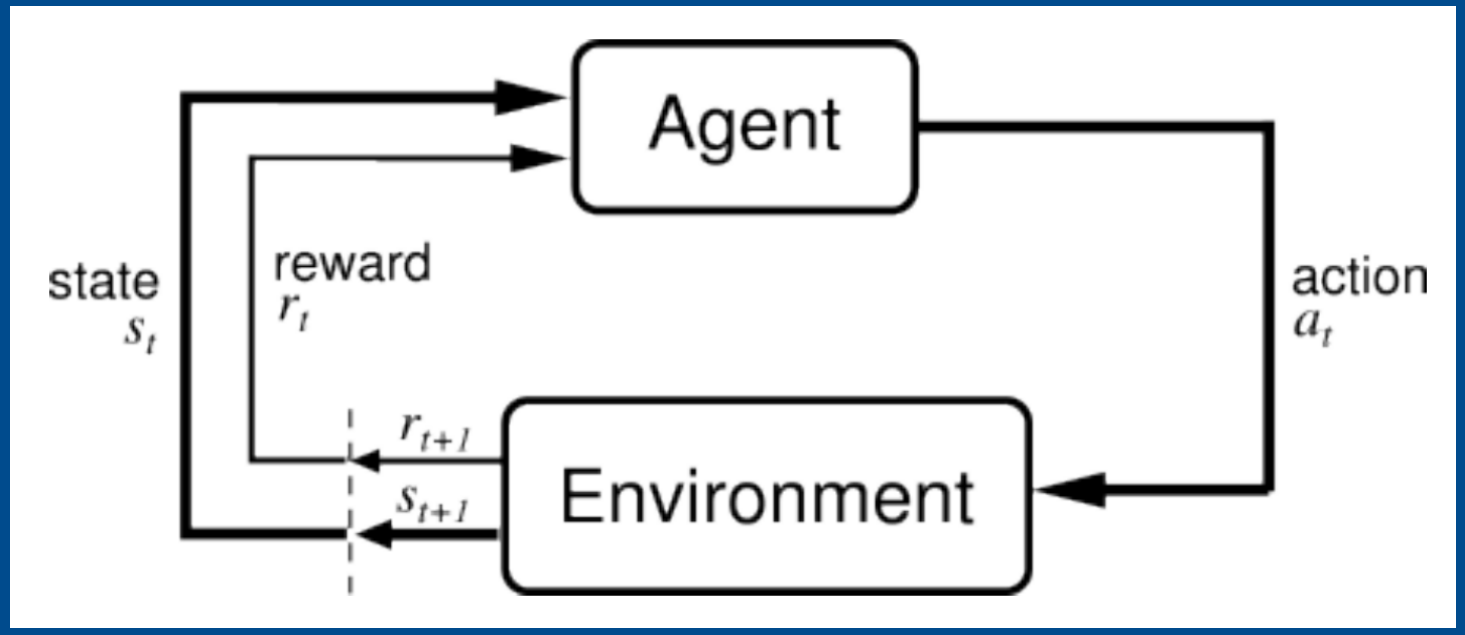
우리 주변에서의 강화

아이가 첫걸음을 떼는 과정도 일종의 강화라고 할 수 있다.

1. 아이는 걷는 것을 배운 적이 없다.
2. 아이는 스스로 이것저것 시도해 보다가 우연히 걷게 된다.
3. 자신이 하는 행동과 걷게 된다는 보상 사이의 상관관계를 모르는 아이는 다시 넘어진다.
4. 시간이 지남에 따라 그 관계를 학습해서 잘 걷게 된다.

강화학습

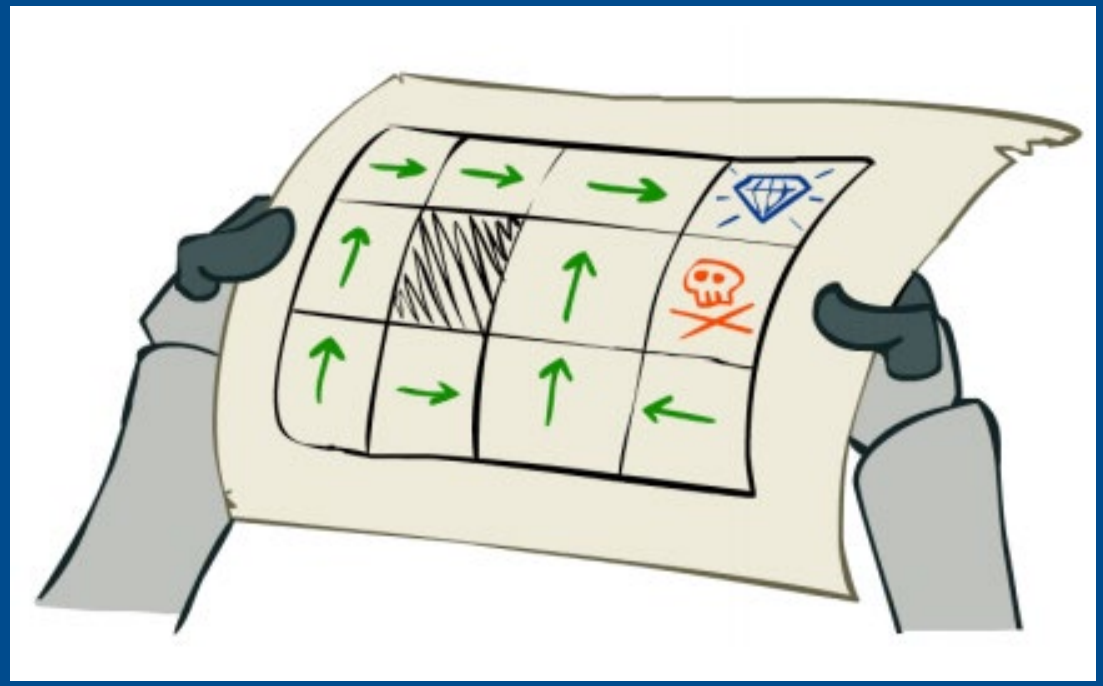
- 에이전트는 사전 지식이 없는 상태에서 학습함
- 에이전트는 자신이 놓인 환경에서 자신의 상태를 인식한 후 행동
- 환경은 에이전트에게 보상을 주고 다음 상태를 알려줌
- 에이전트는 보상을 통해 어떤 행동이 좋은 행동인지 간접적으로 알게 됨



강화학습 문제

결정을 순차적으로 내려야 하는 문제에 강화학습을 적용한다.

이 문제를 풀기 위해서는 문제를 수학적으로 정의해야 한다.



강화학습 문제

수학적으로 정의된 문제는 다음과 같은 구성 요소를 가진다.

1. 상태 (State)
   현재 에이전트의 정보 (정적인 요소 + 동적인 요소)
2. 행동 (Action)
   에이전트가 어떠한 상태에서 취할 수 있는 행동
3. 보상 (Reward)
   에이전트가 학습할 수 있는 유일한 정보, 자신이 했던 행동을 평가할 수 있는 지표
   강화학습의 목표는 시간에 따라 얻는 보상의 합을 최대로 하는 정책을 찾는 것
4. 정책 (Policy)
   순차적 행동 결정 문제에서 구해야 할 답
   모든 상태에 대해 에이전트가 어떤 행동을 해야 하는지 정해놓은 것

MDP

강화 학습은 순차적으로 행동을 계속 결정해야 하는 문제를 푸는 것

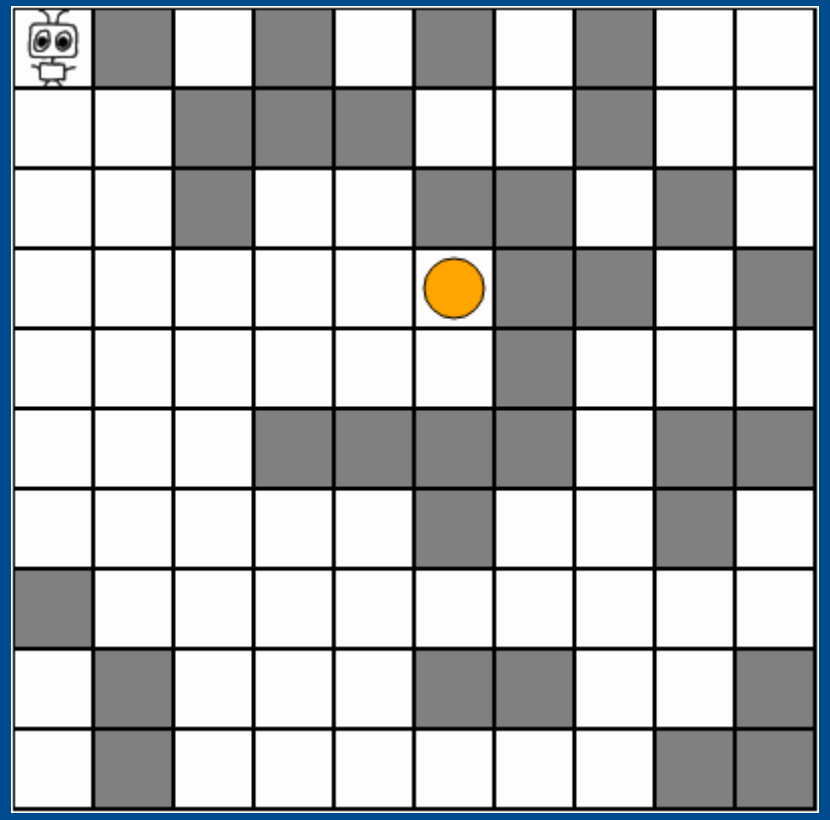
→ 이 문제를 수학적으로 표현한 것이 MDP(Markov Decision Process)

- MDP의 구성 요소

- 상태
- 행동
- 보상 함수
- 상태 변환 확률 (생략)
- 감가율 (생략)

그리드 월드

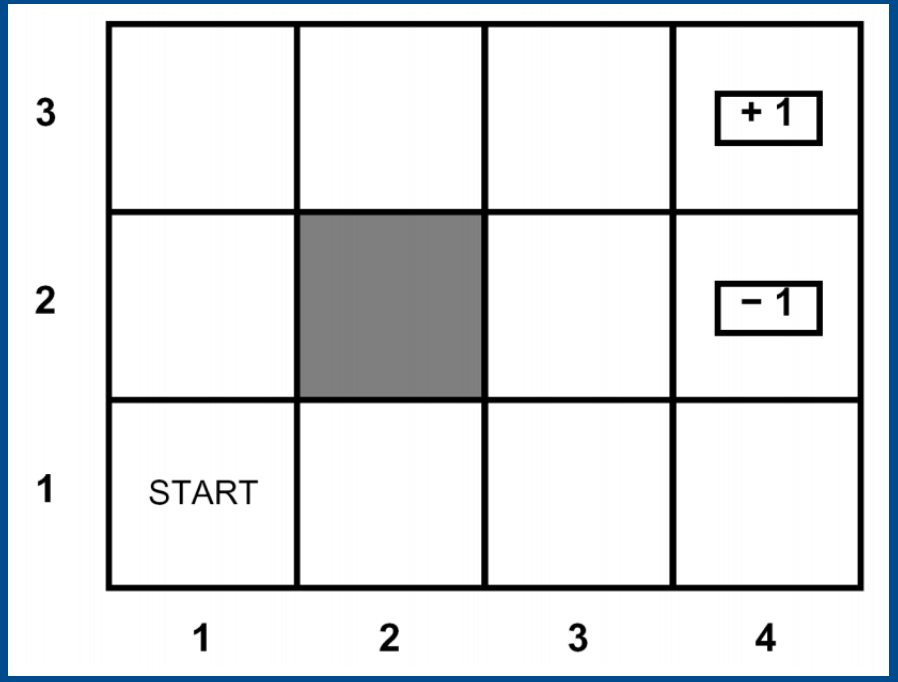
격자로 이뤄진 환경에서 문제를 푸는 각종 예제



상태

에이전트가 관찰 가능한 상태의 집합 : S

- 그리드 월드에서 상태의 개수는 유한
- 그리드 월드에 상태가 5개 있을 경우, 수식으로 표현하면
  $S = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)\}$
- 그리드 월드에서 상태는 격자 상의 각 위치(좌표)
- 에이전트는 시간에 따라 상태 집합 안에 있는 상태를 탐험한다.
  이 때 시간을 t, 시간 t일 때의 상태를 S_t라고 표현한다.
- 예를 들어, 시간이 t일 때 상태가 (1, 3)이라면 $S_t = (1, 3)$



상태

에이전트가 관찰 가능한 상태의 집합 : S

- 어떤 t에서의 상태 S_t는 정해진 것이 아니다.
- 때에 따라서 $t = 1$일 때 $S_t = (1, 3)$일 수도 있고 $S_t = (4, 2)$일 수도 있다.

“상태 = 확률 변수(Random Variable)”

상태

$$S_t = s$$

"시간 t에서의 상태 S_t가 어떤 상태 s다."

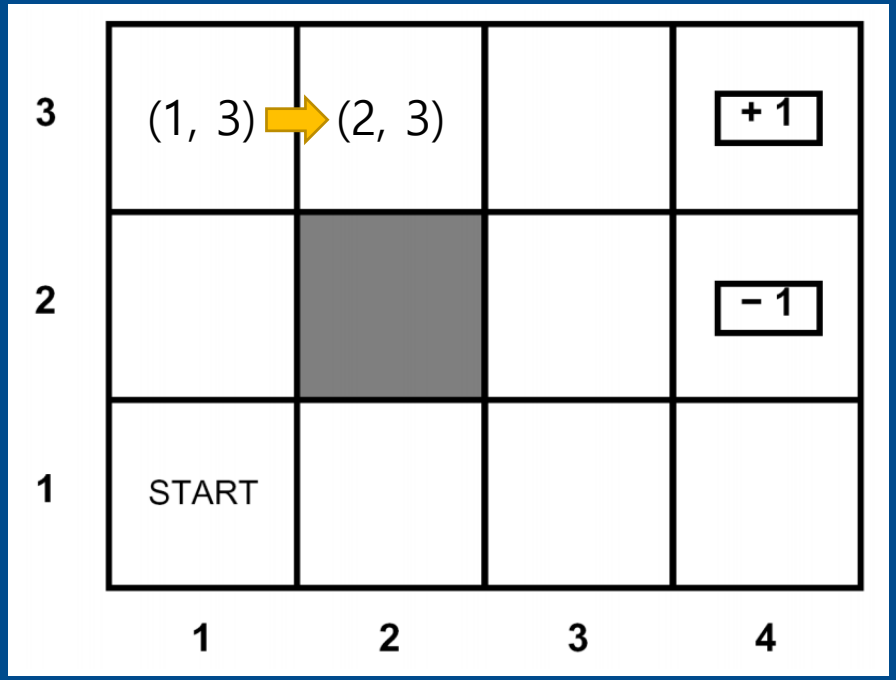
행동

에이전트가 상태 S_t에서 할 수 있는 가능한 행동의 집합 : A

- 보통 에이전트가 할 수 있는 행동은 모든 상태에서 같다.

$$A_t = a$$

- "시간 t에 에이전트가 특정한 행동 a를 했다."
- t라는 시간에 에이전트가 어떤 행동을 할 지는 정해져 있지 않으므로 A_t처럼 대문자로 표현한다.
- 그리드 월드에서 에이전트가 할 수 있는 행동은 $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$
- 만약 시간 t에서 상태가 (1, 3)이고 $A_t = \text{right}$라면 다음 시간의 상태는 (2, 3)이 된다.



보상

에이전트가 학습할 수 있는 유일한 정보

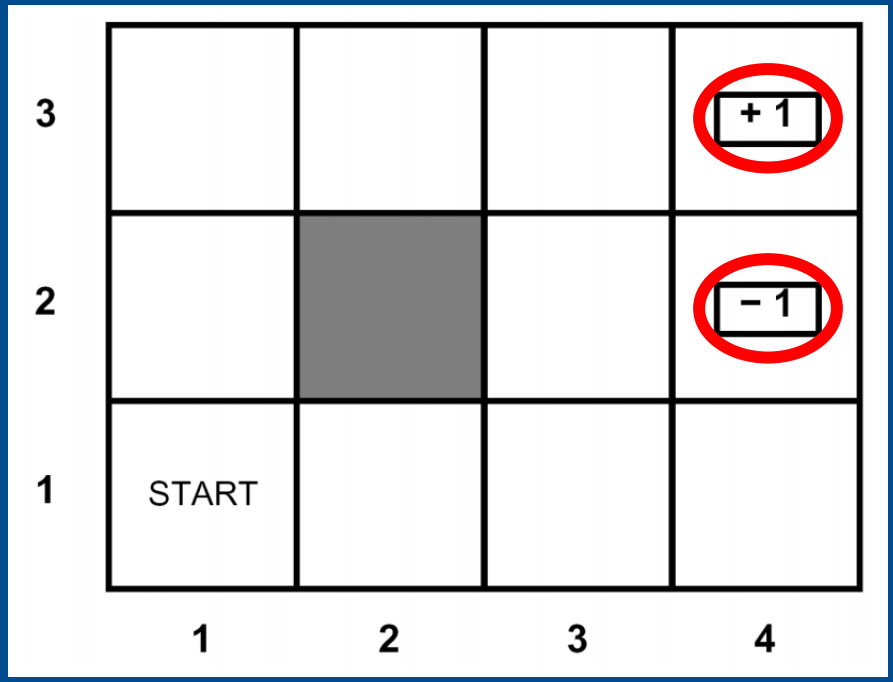
- 보상 함수 (Reward Function)

$$R_s^a = \mathrm{E}[R_{t+1}|S_t = s, A_t = a]$$

- 시간 t일 때 상태가 $S_t = s$이고 그 상태에서 행동이 $A_t = a$를 했을 경우
  받을 보상에 대한 기댓값(Expectation) E
- 에이전트가 어떤 상태에서 행동한 시간 : t
  보상을 받는 시간 : $t + 1$
- 이유 : 에이전트가 보상을 알고 있는게 아니라 환경이 알려주기 때문
  에이전트가 상태 s에서 행동 a를 하면 환경은 에이전트가 가게 되는 다음 상태 s'와
  에이전트가 받을 보상을 에이전트에게 알려준다. 이 시점이 $t + 1$이다.

보상

에이전트가 학습할 수 있는 유일한 정보



정책

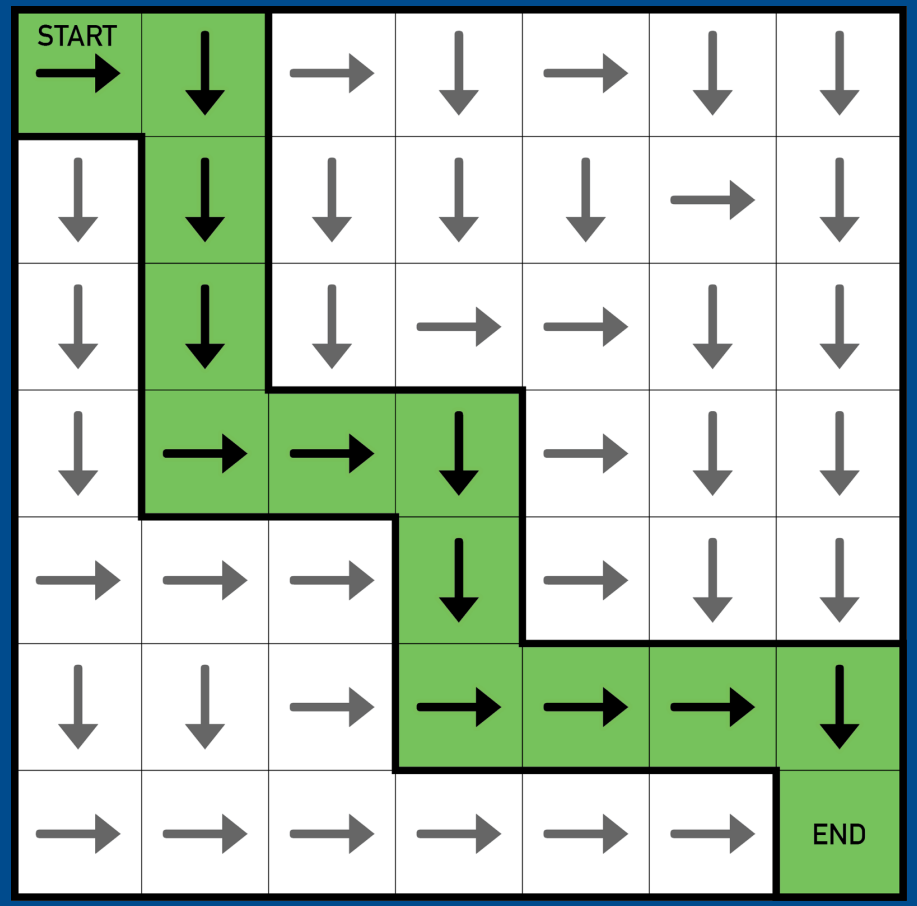
모든 상태에서 에이전트가 할 행동

- 상태가 입력으로 들어오면 행동을 출력으로 내보내는 일종의 함수
- 하나의 행동만을 나타낼 수도 있고, 확률적으로 $a_1 = 10\%$, $a_2 = 90\%$로 나타낼 수도 있다.

$$\pi(a|s) = \mathrm{P}[A_t = a|S_t = s]$$

- 시간 t에 에이전트가 $S_t = s$에 있을 때 가능한 행동 중에서 $A_t = a$를 할 확률
- 강화 학습 문제를 통해 알고 싶은 것은 정책이 아닌 "최적 정책"

정책



가치함수

우리가 지금까지 한 일 : 문제를 MDP로 정의

→ 에이전트는 MDP를 통해 최적 정책을 찾으면 된다.

하지만 에이전트가 어떻게 최적 정책을 찾을 수 있을까?

가치함수

에이전트 입장에서 어떤 행동을 하는 것이 좋은지를 어떻게 알 수 있을까?

→ 현재 상태에서 앞으로 받을 보상을 고려해서 선택해야 좋은 선택!

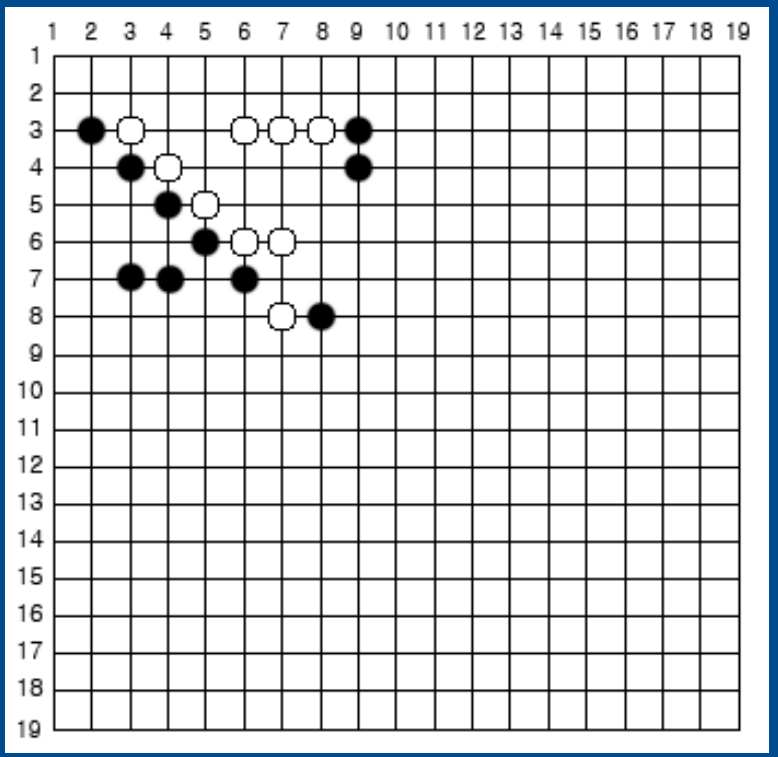
하지만 아직 받지 않은 보상들을 어떻게 고려할 수 있을까?

→ 에이전트는 가치함수를 통해 행동을 선택할 수 있다.

오목 게임 만들기

2020 OSS Winter
AlphaZero 오목 AI - Day 1

- 바둑판에 검은 돌과 흰 돌을 교대로 놓아서 겨루는 게임
- 바둑판에는 19개의 가로줄과 19개의 세로줄이 있다
- 같은 색 돌을 연속해서 5개 놓으면 이긴다
  - 여기서 연속적이란 가로, 세로 또는 대각선 방향 모두를 뜻함
- 하지만 6개 이상을 연속해서 놓은 경우에는 이긴 것이 아니다 (오목이라고 할 수 없다)



오목 게임 만들기

이제 오목 게임을 만들어 보자. 만들 파일은 다음과 같다.

- types.py : 플레이어, 바둑판 내 위치, 방향 등의 타입이 저장된 파일
- board.py : 바둑판, 행동, 게임 상태 등이 저장된 파일
- utils.py : 바둑판, 행동 출력 및 좌표 변환 함수가 저장된 파일
- zobrist.py : Zobrist 해시 값들이 저장된 파일

간단한 오목 AI 봇 만들기

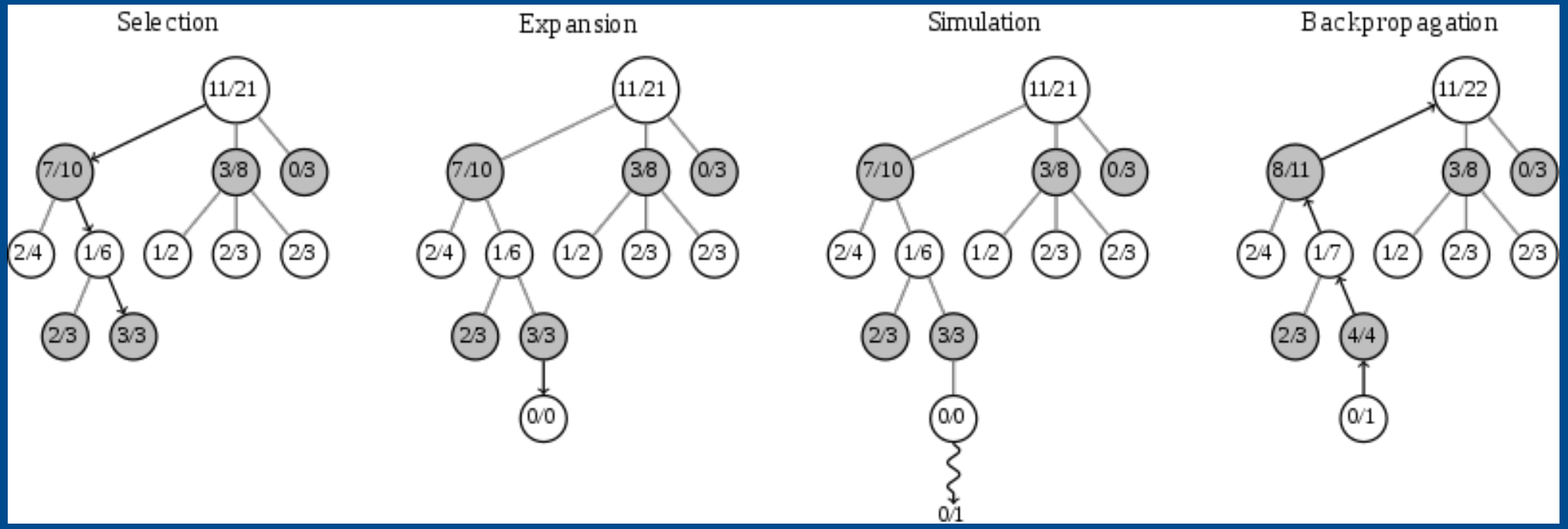
이제 오목 게임을 하는 간단한 AI 봇을 만들어 보자.

- agents.py : 기본 클래스에 해당하는 에이전트가 저장된 파일
- naive.py : 임의의 위치에 돌을 놓는 에이전트가 저장된 파일
- bot_v_bot.py : AI 봇 vs AI 봇 오목 게임을 할 수 있는 파일
- human_v_bot.py : 인간 vs AI 봇 오목 게임을 할 수 있는 파일

MCTS

Monte Carlo Tree Search

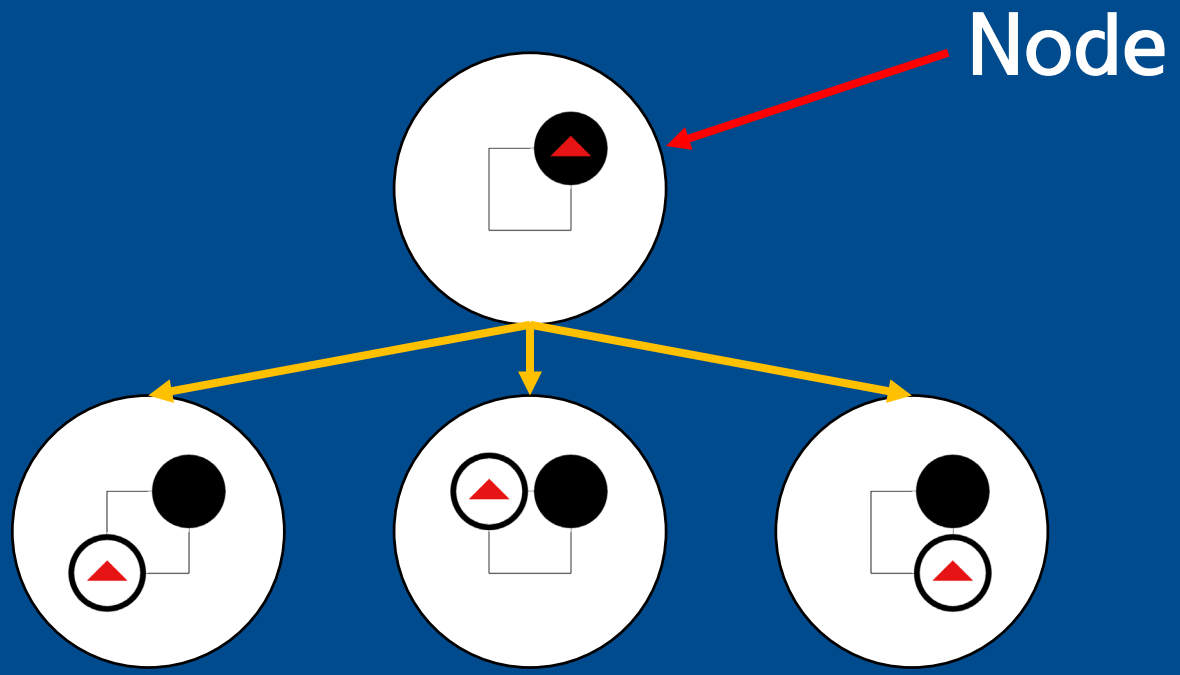
→ 모종의 의사 결정을 위한 트리 탐색 알고리즘. 검색 공간에서 무작위 추출에 기초해 트리를 확장한다.



MCTS

Tree 자료 구조

: 게임의 진행 상황을 저장하는 구조



MCTS

노드(Node)는 트리에서 한 시점의 게임 상태를 의미한다.

노드는 다음의 값들을 갖는다.

- $N(s,a)$: 노드를 탐색한 횟수
- $W(s,a)$: 이 행동을 했을 때 이긴 게임의 개수

MCTS

MCTS엔 4가지 단계가 있다.

- 선택(Selection) : 현재 게임 상태가 트리 내에 존재하는 동안 다음에 수행할 행동을 선택한다. 이 때 활용(Exploitation)과 탐험(Exploration)의 균형을 맞춰 저장된 통계 값에 따라 선택한다.
- 확장(Expansion) : 현재 게임 상태가 트리 내에 존재하지 않으면 새로운 노드로 확장한다.
- 시뮬레이션(Simulation) : 게임이 끝날 때까지 다음에 수행할 행동을 임의로 선택한다.
- 역전파(Backpropagation) : 게임을 실행한 노드로부터 루트 노드까지 통계 값을 갱신한다. (여기서 통계 값이란 총 게임 실행 횟수, 승리 횟수 등을 말함)

MCTS - Selection

Root 노드부터 시작해 트리의 가장 밑바닥까지 탐색할 수를 선택하는 과정.

다음 수식에 따라 수를 선택하고, 이 과정을 트리의 바닥에 도달할 때까지 반복

$$a_t = \mathrm{argmax}_t \left(\frac{W(s,a)}{N(s,a)} + c_\tau \sqrt{\frac{\log \sum_b N(s,b)}{N(s,a)}} \right)$$

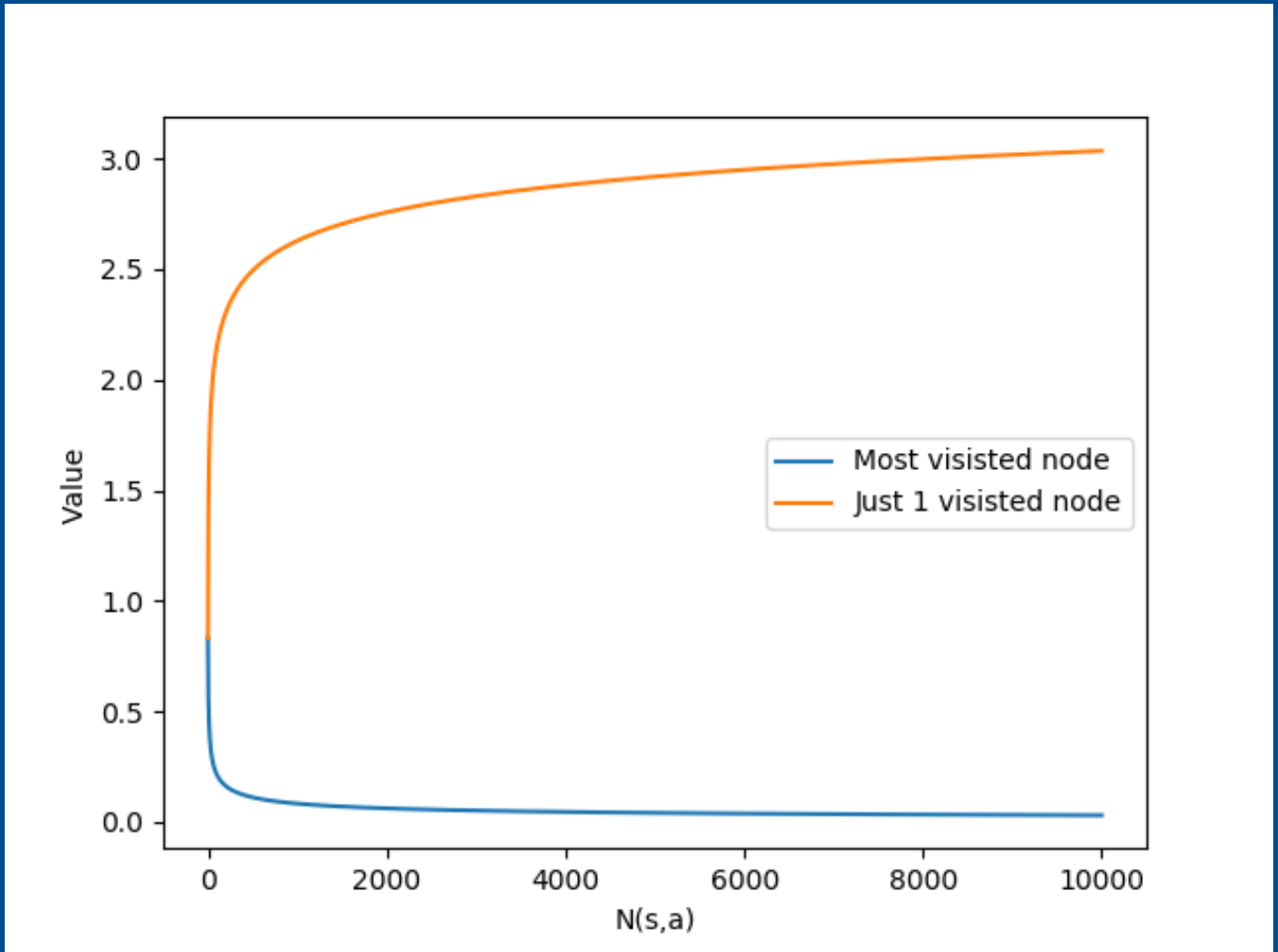
MCTS - Selection

앞의 수식에서 $\frac{W(s,a)}{N(s,a)}$는 해당 행동의 승률이다.

→ 승리할 확률이 가장 높은 행동을 우선적으로 선택한다.

$c_\tau\sqrt{\frac{\log\sum_b N(s,b)}{N(s,a)}}$ 은 특정 수만 탐색하는 걸 방지하기 위한 항이다.

MCTS - Selection



MCTS - Selection

c_τ는 탐색 상수로, 트리의 깊이와 넓이를 조절한다.

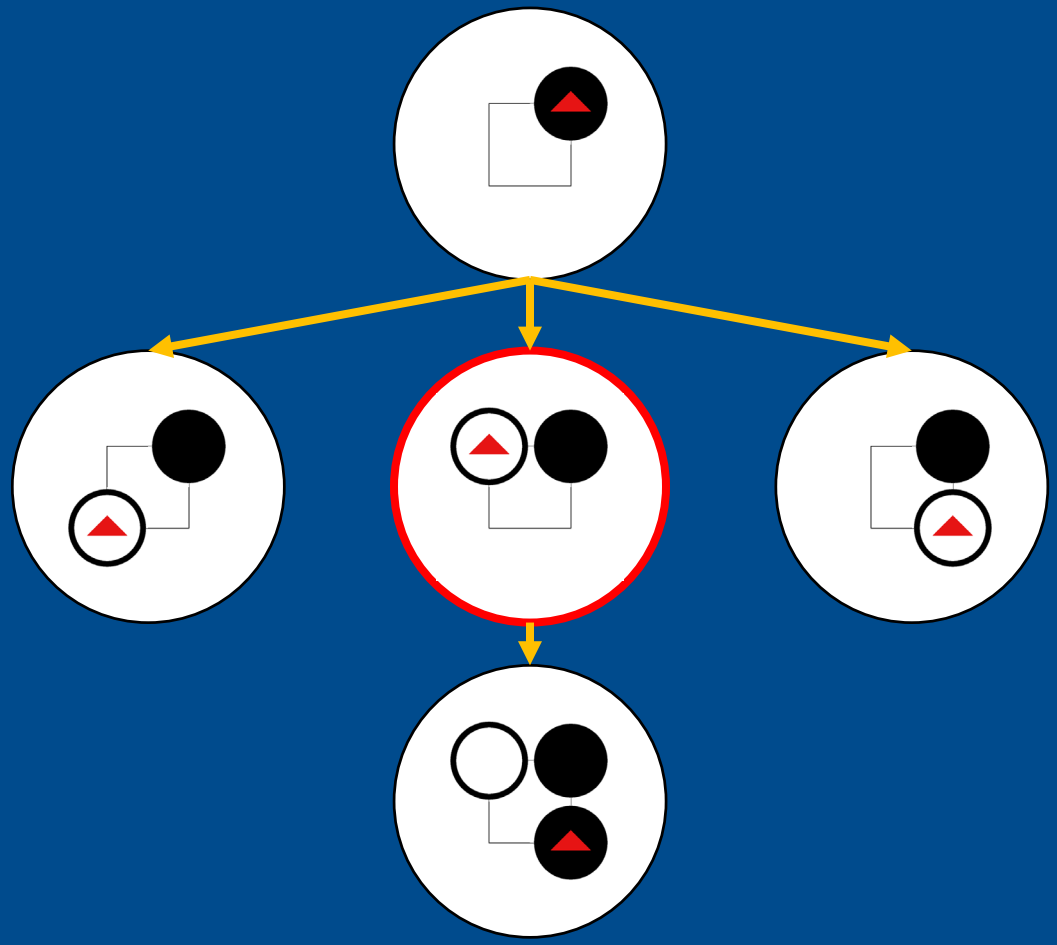
- 만약 c_τ가 높다면 $c_\tau\sqrt{\frac{\log\sum_b N(s,b)}{N(s,a)}}$의 영향이 커진다.
  → 트리가 넓어진다.

- 만약 c_τ가 낮다면 $c_\tau\sqrt{\frac{\log\sum_b N(s,b)}{N(s,a)}}$의 영향이 작아진다.
  → 트리가 깊어진다.

※ 넓이와 깊이 사이의 적절한 균형을 맞추는 게 중요하다.

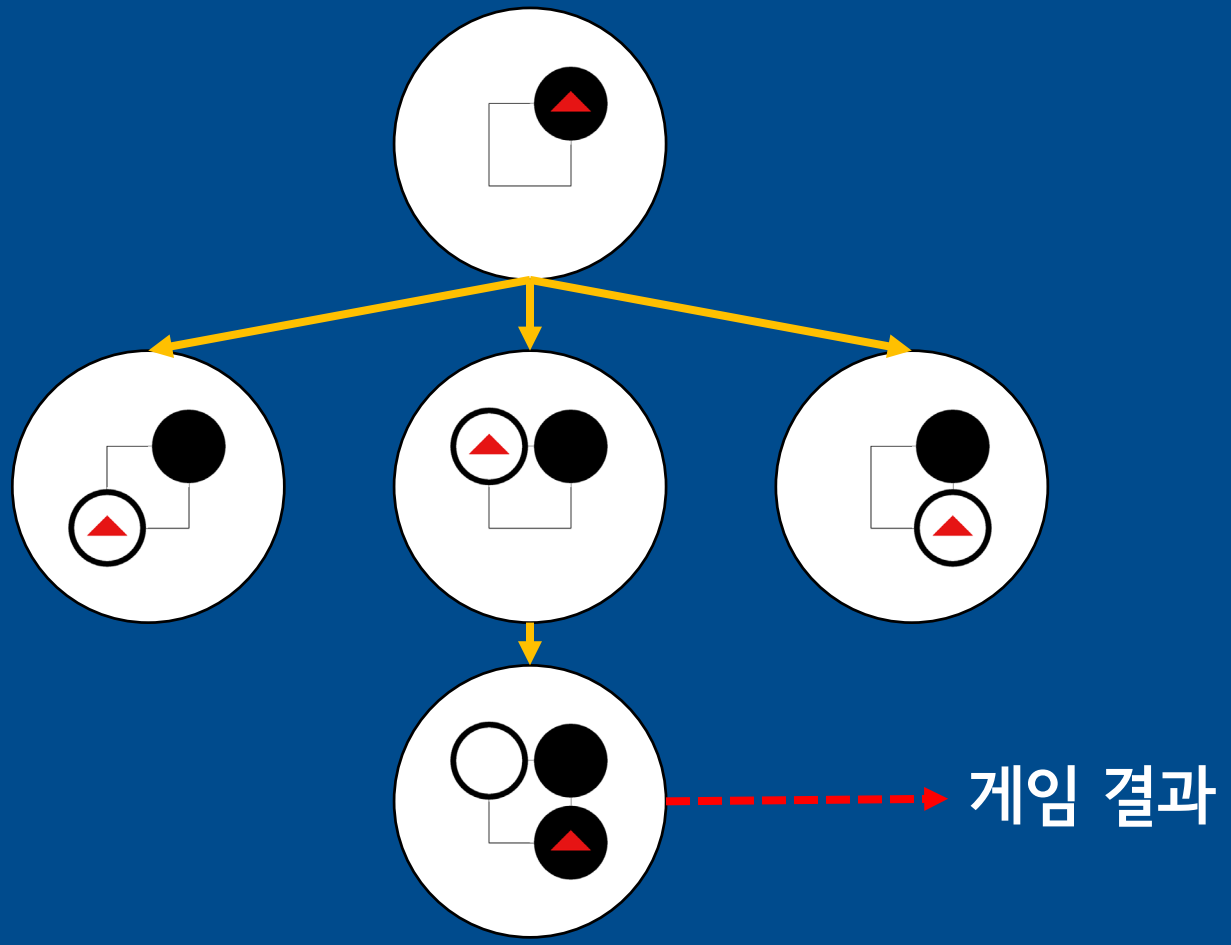
MCTS - Expansion

Selection이 끝나면 그 다음 상태를 트리에 넣어 트리를 확장한다.



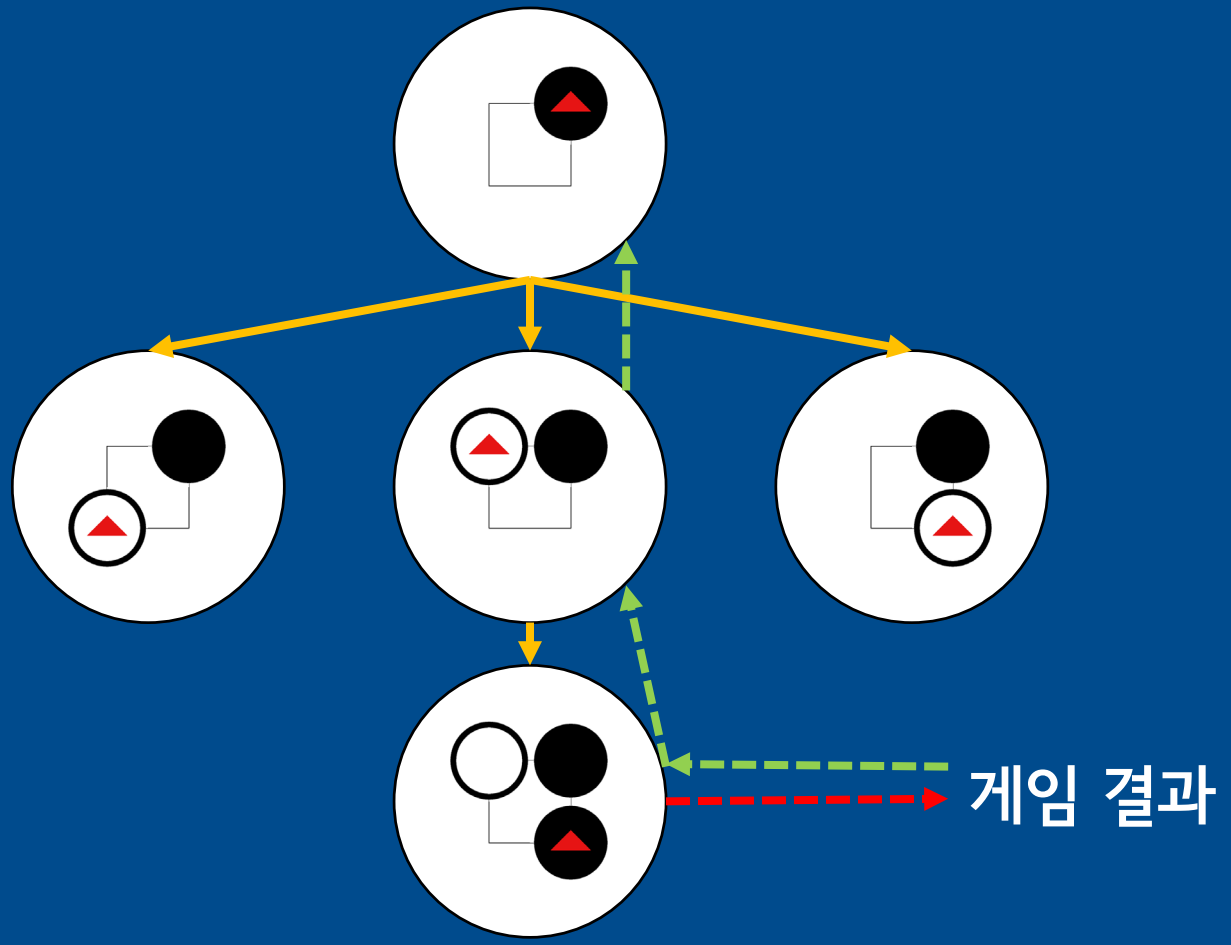
MCTS - Simulation

확장된 노드로부터 게임을 끝까지 무작위 플레이한다.



MCTS - Backpropagation

게임의 결과를 바탕으로 트리를 거슬러 올라가며 노드의 값을 갱신한다.



MCTS - Playout

Playout은 선택, 확장, 시뮬레이션, 역전파를 1회 수행하는 것이다.

MCTS는 무한히 돌리면 최적해(Optimal Solution)에 수렴한다.

하지만, MCTS를 무한히 돌릴 수 없어 playout에 다음 등의 제한을 둔다.

- Playout 수 제한
- 탐색 소요 시간 제한

감사합니다

http://github.com/utilForever

질문 환영합니다!