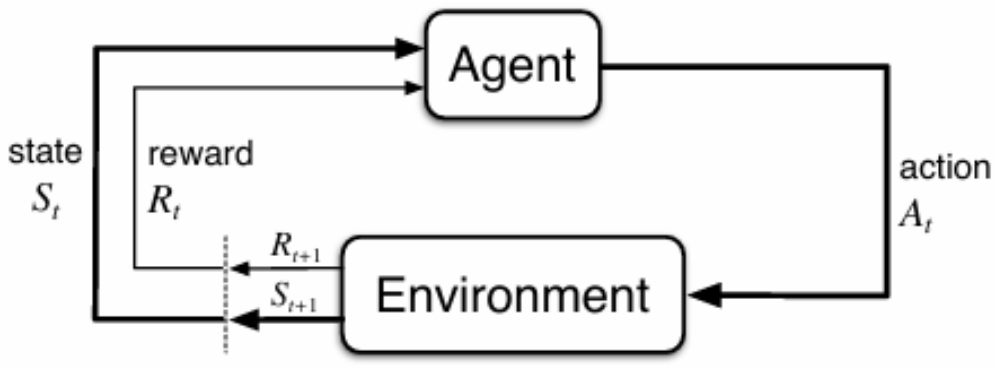
Reinforcement Learning

강화학습(Reinfocement Learning)

- 순차적 선택을 잘 하도록 학습하여 인공지능이 직접 일련의 선택을 하도록하는 학습 방법 => 시행착오를 통해 최적의 결과를 도출하는 행동의 시퀀스 선택
- Agent와 Environment의 상호작용으로 학습
- Agent -> action -> Environment(환경 상태 변화)
  - Time t에 state와 reward를 받아 action 수행
- Environmnet -> reward -> agent(행동에 대한 보상 제공)
  - Time t에 action을 받고 환경 변화, 이후 t+1의 state와 reward 제공



Terminology

- State: environment의 현재 상태에 대한 모든 정보(vector, matrix, tensor 등)
- Action Spaces: 주어진 환경에서 가능한 actio의 set
  - Discrete Action Space: action의 개수가 유한한 경우
  - Continous Action Space: action의 개수가 무한한 경우
- Policy: Agent가 어떤 action을 취할지 선택하는 정책
  - Deterministc Policy: 각 state에서 취할 action이 정해져 있는 경우
  - Stochastic Policy: 각 state에서 취할 actio이 정해져 있지 않고 확률적인 경우
- Reward & Retruns: S_t와 A_t를 받아 environment가 agent에게 제공하는 기대값

Ex)Tic Tac Toe 문제 정의

- 게임 규칙

- 3 x 3 보드, 두 명의 플레이어가 번갈아 가며 착수
- 가로, 세로, 대각선으로 한 줄을 먼저 채우면 승리
- 게임의 최대 길이: 9(max length, max depth)

- 학습 문제 정의

- State: 현재 보드의 상태(빈칸, 흑돌, 백돌)
- Action: 가능한 착수 위치에 돌을 두는 것
- Reward: 승리 +1, 패배 -1, 무승부 0 (보편적)
- Environment: 보드

Tic Tac Toe 문제 정의

- 9곳 착점 가능, 전체 게임 복잡도 $3^9 = 19{,}683$
- 흑이 최대 대국자, 백이 최소 대국자라고 했을 때
  - 첫 대국자인 흑이 중앙에 둔 경우 게임 복잡도는 $3^8 = 6{,}561$
  - 불가능한 경우 -> 총 4,724; 정상적인 패턴의 수 1,837개
  - 회전과 상하 대칭 -> 1,582개; 원형 패턴의 수 255개

		depth 1	depth 2	depth 3	depth 4	depth 5	depth 6	depth 7	depth 8	depth 9	total
Normal patterns	Black win	0	0	0	0	60	0	264	0	36	360
	White win	0	0	0	0	0	36	0	48	0	84
	Draw	0	0	0	0	0	0	0	0	8	8
	Not finished	1	8	56	168	360	444	256	92	0	1,385
	Total	1	8	56	168	420	480	520	140	44	1,837
Base cases	Black win	0	0	0	0	12	0	35	0	6	53
	White win	0	0	0	0	0	5	0	6	0	11
	Draw	0	0	0	0	0	0	0	0	1	1
	Not finished	1	2	8	24	48	59	35	13	0	190
	Total	1	2	8	24	60	64	70	19	7	255

Tic Tac Toe 문제 정의

- 게임 결과에 따른 reward 함수

$$r(t) = \begin{cases} +2^t & for\ win \\ -2^t & for\ lose \\ \dfrac{\sin(t)}{t} & for\ draw \end{cases}$$

- L : 게임 길이
- T = 9 - l

=> 게임 길이가 길면 작은 보상값, 짧으면 큰 보상값 부여

0	1	2
3	4	5
6	7	8

.376	.218	.360
.190	.506	.192
.361	.198	.341

몬테카를로 방법(MC)

- 난수를 이용해 함수 값을 확률적으로 계산하는 알고리즘
- 계산하려는 값이 수학적 형식으로 표현되지 않거나, 복잡한 경우에 근사적인 값을 계산하기 위해 사용
- 임의 추출(random sampling)을 이용해 계산(random simulation -> 평균 결과 계산)

✓ 실시간으로 최선의 수를 선택해야 하는 경우에는 적용이 어려움
✓ 게임이 끝날 때까지 기다려야 함

MCTS(Monte-Carlo Tree Search)

- Position evalution function을 필요로 하지 않는 best-first search(최대 우선 탐색 방법)
- 탐색 공간에 대해 무작위적 탐험을 근간으로, 이전의 탐험 결과를 사용해 메모리에 점진적으로 게임트리 구축→ 시뮬레이션을 통해 착수 계산

∴ 점진적으로 승리 우위를 가질 수 있는 착점을 기준으로 게임 트리를 구성
반복적인 무작위 샘플링을 기반으로 하는 통계적 알고리즘
게임트리의 노드들을 전역탐색해야 하는 단점 해결

† Position evaluation function: 현재 상태 평가 -> 유리한 정도를 점수로 나타냄

승/패 결과만을 이용해 최적의 수를 찾아가는 MCTS는 해당 함수는 필요하지 않음

MCTS(Monte-Carlo Tree Search)

▪ MCTS 적용 조건

- 득실값(payoff)의 한계가 있어야 함
  - ➢ 게임의 최대/최소 점수값 존재

- 게임 규칙이 정해져 있는 완전 정보 게임이어야 함

- 게임의 길이가 제한되어 있으며, 비교적 빨리 시뮬레이션이 끝날 수 있어야 함

MCTS(Monte-Carlo Tree Search)

▪ 탐색 트리(Searh tree)

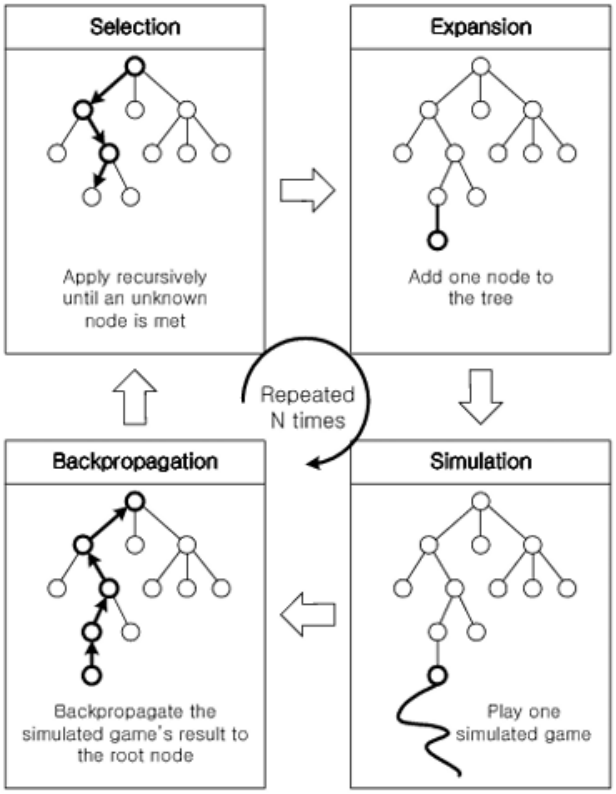
- 시뮬레이션을 위해 사용되는 정보를 담은 트리
- MCTS가 탐색하는 현재까지 학습된 정보 저장(현재까지 탐색한 부분만 포함)
- 특정 상황에서 승률을 알고 싶은 노드 = 탐색 트리의 단말노드 <MCTS가 계산할 가치의 기준점>

▪ 게임트리(Game tree)

- 게임 진행 중에 생성된 착점들에 대한 정보를 담은 트리
- 가능한 모든 상태를 포함하는 이론적 트리로, 완전한 게임 시뮬레이션 트리
- 탐색 트리에서 현재 노드가 종료상태가 아니라면 게임트리의 단말 노드가 나올 때까지 랜덤 시뮬레이션 수행
- 게임트리의 단말 노드 = 게임이 완료된 상태(승/패/무승부 결정)

MCTS(Monte-Carlo Tree Search)

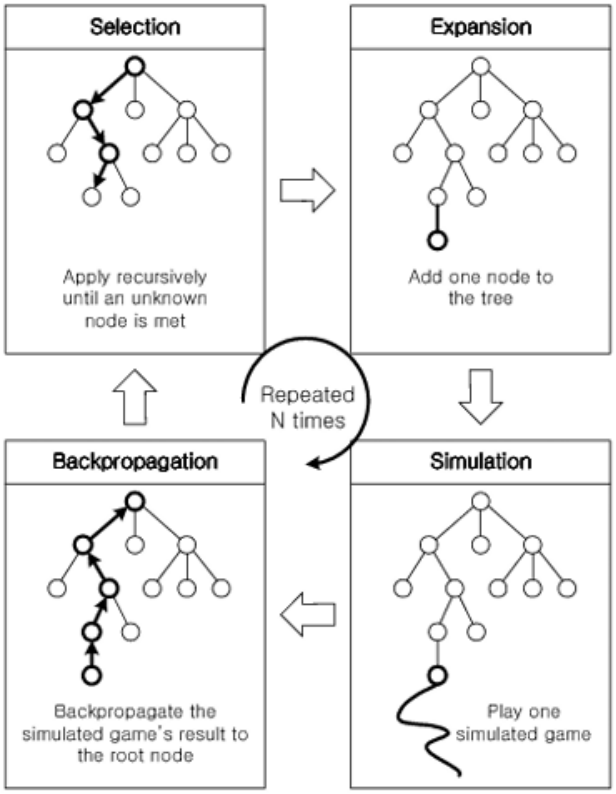
- **탐색트리에서 MCTS 알고리즘**



1. Selection: Selection function을 탐색 트리의 단말 노드를 만날 때까지 반복적으로 적용
2. Expansion: 선택 단계에서 선택된 하나 이상의 노드의 자식을 탐색트리에 첨가
3. Simulation: 탐색 트리의 단말노드에서 게임 트리의 단말 노드를 만날 때까지 수행
4. Backpropagation: 시뮬레이션의 결과로 얻은 보상값을 선택된 노드에서부터 탐색트리의 뿌리 노드까지 역전파

MCTS(Monte-Carlo Tree Search)

- **기존 방식에서 발생할 수 있는 문제**



반복적 무작위 탐험에 의해 계산된 통계적 확률값에 의존

⇒ 계산된 수학적 확률이 같은 경우 원하는 곳에 착수하지 않을 수 있음

⇒ 두 번째 대국자인 경우 전체 게임에 대한 최소 평균 승률값을 선택하게 됨

∴ 첫 번째 대국자인 최대 대국자에게 최선의 대응을 할 수 없는 문제 발생

연속되는 게임의 행위를 어떤 방식으로 시뮬레이션 해야 하는지 알 수 없음

S-MCTS(Strategic Monte-Carlo Tree Search)

- **전략적 몬테카를로 트리 탐색**

```
function MCTS(Position, NoOfSimulations)
  for each child n available from Position
    reward[n] = 0
    for i from 1 to NoOfSimulations
      POS = copy(Position)
      while (POS is terminal node)
        MCTS(POS, random child from POS)
      end while
      if depth == 2 then
        reward[n] += 2 * result(POS)
      else reward[n] += result(POS)
    end for
    reward[n] /= NoOfSimulations
  end for
  return(child i with highest reward[i])
end function
```

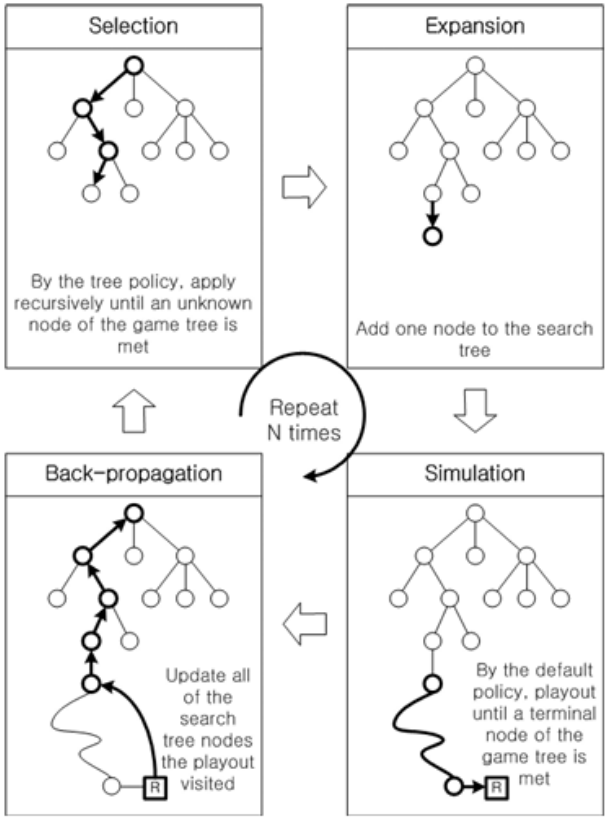
수학적 확률값이 최대치인 곳 대신

전략적으로 더 급한 곳을 착수할 수 있도록 개선

(두 번의 착수 후에 지는 곳을 우선 봉쇄)

UCT algorithm(upper Confidence Bound applied to Trees)

- **최상의 첫 수를 두기 위한 신뢰상한 트리 알고리즘**



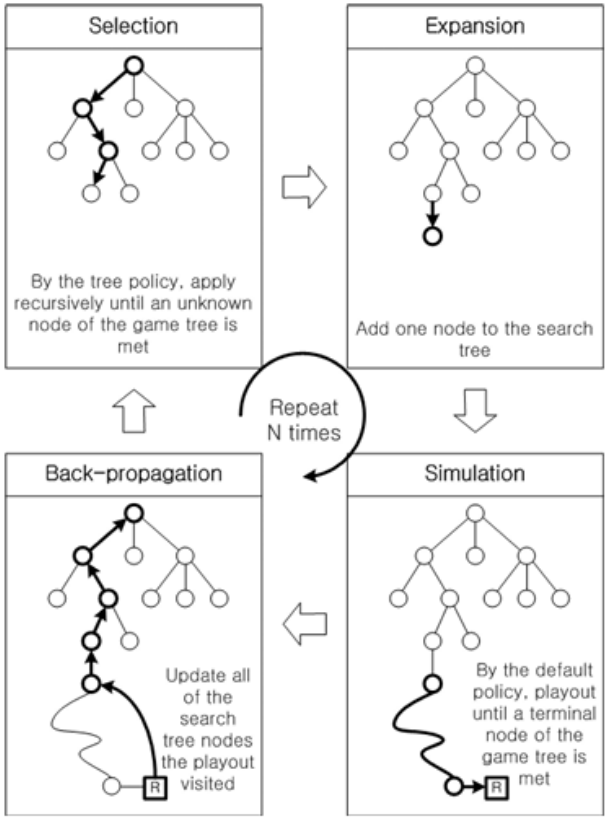
- MCTS의 변형
- 신뢰상환(Upper Confidence Bounds) 정책을 따라 행동 결정

⇒ Selection 단계에서 트리 정책을 위해 UCB 사용

⇒ Simulation 단계에 있는 디폴트 정책을 위해 MC 시뮬레이션 사용

UCT algorithm(upper Confidence Bound applied to Trees)

- **최상의 첫 수를 두기 위한 신뢰상한 트리 알고리즘**



1. Selection: 트리 정책을 반영하여 게임트리 내 확장이 안 된 노드들이나 단말 노드를 만날 때까지 반복하여 노드 선택
2. Expansion: 탐험이 안 된 노드를 탐색 트리에 추가
3. Simulation: default 정책에 의해 수행되어 탐색 트리의 단말노드에서 게임 트리의 단말 노드를 만날 때까지 수행
4. Backpropagation: 시뮬레이션의 결과로 얻은 보상값을 선택된 노드에서부터 탐색트리의 뿌리 노드까지 역전파

UCT algorithm(upper Confidence Bound applied to Trees)

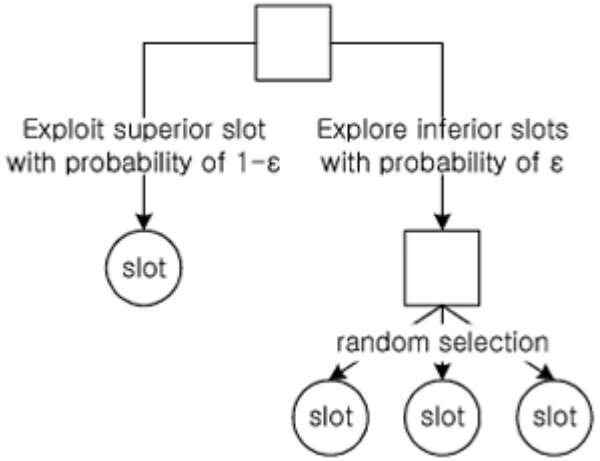
Backgrounds

- 다중슬롯머신(MAB: Multi-Armed Bandit) 문제
  - 취할 수 있는 행동 多
  - 각각의 행동을 통해 얻게 될 보상 정보 불완전
  - 위와 같은 조건 아래서 보상을 극대화하기 위해 어떤 행동을 해야 될지 추론하는 수학적 모델

  - Non-Stationary Reward -> 보상의 분포가 시간의 흐름에 따라 변화한다면?
- 탐험-활용 딜레마(Explore-Exploit dilemma)
  - 보상값이 높은 행동에 더 많은 시간 투자를 하기 위해
  - 탐험을 통해 더 좋은 행동을 찾고자 하기 위해
  - 탐험과 활용의 적합한 선택을 찾고자 하는 문제

Explore-Exploit dilemma

- **Epsilon-greedy algorithm**
  - ϵ만큼 탐험 수행, 1-ϵ만큼 활용 수행
  - 가장 성공률이 높은 행동을 기본으로 수행, 가끔씩 무작위 행동 수행
  - 엡실론 값을 인위적으로 고정 -> 강제적 탐험 발생



Explore-Exploit dilemma

Selection 단계에 적용되는 UCB 알고리즘

$$UCB1 = \bar{X}_i + C\sqrt{\frac{2\ln n}{n_i}}$$

- $\bar{X}_i$: 현재 검사된 노드 i로부터의 평균 보상값(활용부)
- C: Exploratin bias(탐험부)
- n: i노드의 부모노드를 방문한 횟수(탐험부)
- n_i: 노드 i를 방문한 횟수(탐험부)

⇒ 수행해 왔던 행동의 유용도 판단 → 활용과 탐험의 균형 유지

⇒ 행동의 선택에 있어 현재 가장 높은 UCB 값을 갖는 행동

⇒ 가끔씩 탐험 결정(현재 최적처럼 보이지 않는 행동 시행)

⇒ 시간이 지남에 따라 UCB 값이 가장 높은 행동을 많이 시행

Tic Tac Toe 문제 해결

- Minimax 적용 한계

• 상대방이 특정한 방법으로 게임을 한다는 가정

⇒ 현실에서 상대방이 항상 최적의 선택을 하지 않을 수 있음

⇒ 잠재적으로 승리할 수 있는 게임 상태임에도 불구하고, 상대의 비효율적인 플레이를 고려하지 않아 해당 상태(승리로 끝나는 마지막 state)에 도달하지 못할 수도 있음

- DP 적용 한계

• 상대방이 각 보드 상태에서 어떤 선택을 할 확률까지 알고 있어야 함(비현실적)

• 실제 문제에서는 상대방의 플레이 스타일을 사전에 알 수 없는 경우가 대부분

Tic Tac Toe 문제 해결

- MCTS 적용 한계

• 랜덤 플레이어 vs 랜덤 플레이어

⇒ 여러 번 랜덤하게 시뮬레이션을 돌려 보고 가장 승률이 높은 수 선택

⇒ 최적의 전략을 항상 찾지 못할 수 있음

⇒ 최적의 수를 보장하지 못함

Tic Tac Toe 문제 해결

- 강화학습(Reinforcement Learning)
- 상대방이 어떤 전략을 사용할지 경험을 통해 학습
  - 직접 게임을 하며 데이터를 모아 학습
- 상대방과 여러 번 대결하며, 상대방 행동의 모델을 신뢰 수준까지 학습한 후 대략적인 상대 모델을 고려
- 행통패턴 학습(상대 전략) -> 전략(정책_ 최적화)

<-> minimax; 상대가 실수하는 경우에도 대처 가능

<-> MCTS; 랜덤한 탐색을 줄이고 빠르게 최적 전략 적용 가능

∴ Intermediate Reward가 잘 설정될 수 있도록 학습하는 게 중요!

MDP

- 마크로프 결정과정
- 구성 요소
  - 상태(S): 에이전트가 관찰 가능한 상태의 집합
  - 행동(A): 상태 S_t에서 취할 수 있는 행동
  - 상태 변환 확률(P): 어떤 변수에 의해 의도한 상태와 다른 상태로 바뀌는 것
  - 보상 함수(R): 에이전트가 학습할 수 있는 정보로, 환경이 제공
  - 감가율(γ): 0~1 사이의 수, 보상의 가치 결정

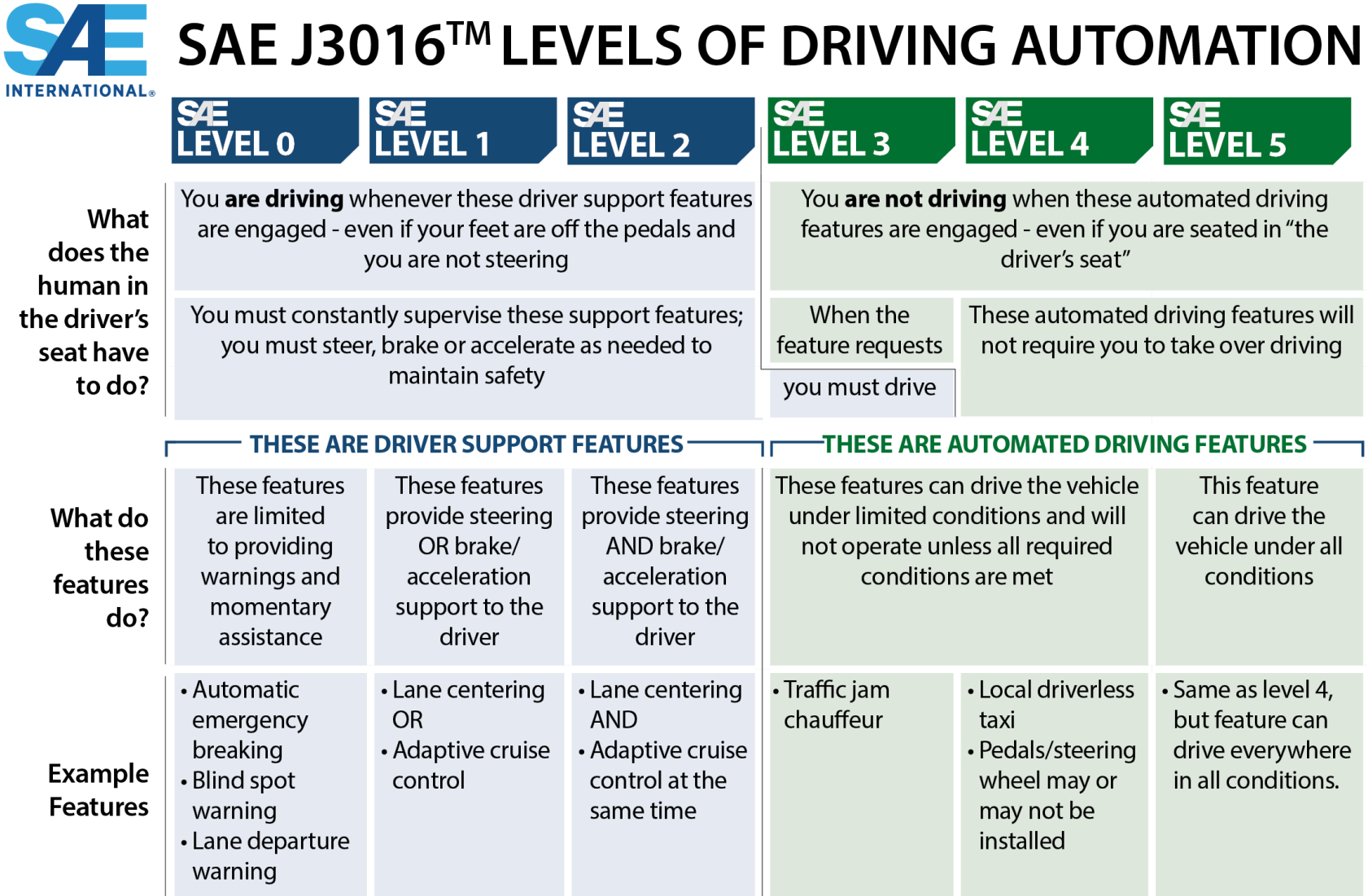
$\Rightarrow$ 확률적으로 변하는 것들을 모델링

$$P_{ss'}^{a} = P[S_{t+1} = s \mid S_t = s, A_t = a] \qquad r(s,a) = E[R_{t+1} = s \mid S_t = s, A_t = a]$$

Def

- 벨만 방정식 (Bellman Equation)
- 현재 상태의 가치 함수와 다음 상태의 가치 함수 사이의 관계를 식으로 나타낸 것
  - 정책(Policy): 에이전트가 어떤 상황에서 취할 행동(조건부 확률)
    - 어떤 상태 s에서 행동 a를 취하는 확률 p
    - $\pi(a|s) = [A_t = a \mid S_t = s]$
  - 가치 함수(Value Function): 받을 수 있는 보상을 함수로 표현
    - State value function: 어떤 상태 s에서 받는 보상
      - $v_\pi(s) = E_\pi[G_t \mid S_t = s]$
    - Action value function: 어떤 상태 s에서 행동 a를 했을 때 받는 보상
      - $q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$
  - 시점 t부터 현재 시점 k까지 받은 보상
    - $G_t = R_{t+1} + r^1 R_{t+2} \cdots + r^{k-1} R_{t+k}$

Example:Autonomous Vehicle



Example:Autonomous Vehicle

- Scene representation을 위한 sensor의 역할(environment 구성)

카메라 (RGB/IR)	도로, 교통신호, 보행자 인식
LiDAR (360° / 단일)	3D 환경 매핑, 거리 및 깊이 감지
RADAR (LRR/SRR)	속도 및 거리 측정, 차량 및 장애물 감지
Ultra Sensor	근거리 장애물 감지, 주차 보조
GPS + IMU	위치 추적 및 차선 유지

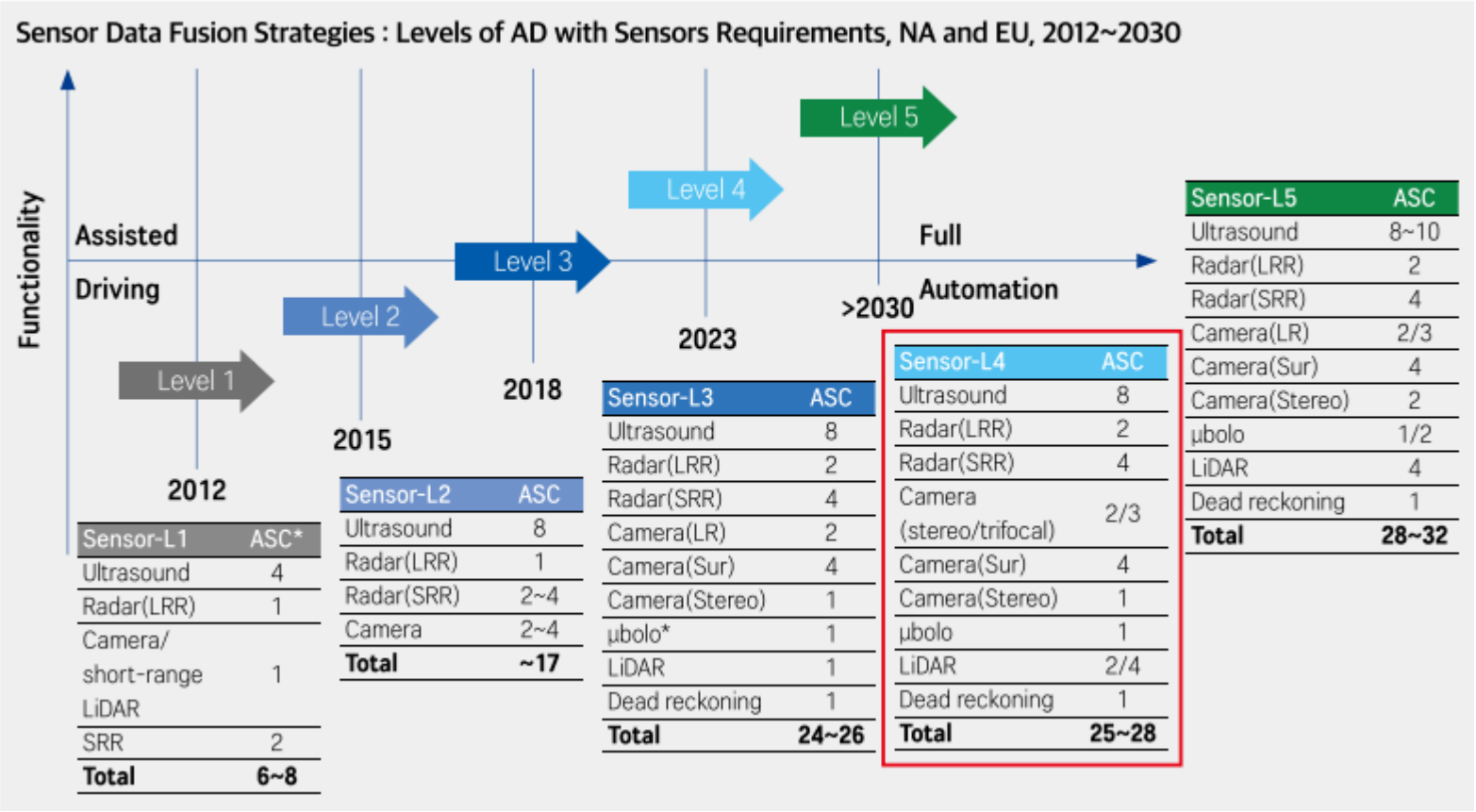
Example:Autonomous Vehicle

- **SAE 레벨 별 spec**

- Level 2(ex. Google Car(2009)): 1 Camera, 1 Radar, 1 360° LiDAR
- Level 2~3(ex. Tesla Model X(2016)): 8 RGB Cameras, 1 Radar, 12 Ultrasonic
- Level 3~4(ex. Uber ATG(2018)): 7 RGB Cameras, 1 360° LiDAR, 10 Radars
- Level 4(ex. Waymo 5^{th} Gen(2019)): 29 RGB Cameras, 4 LiDAR, 5 Radars
- Level 4~5(ex. Tesla Model 3): 29 RGB Cameras, 4 LiDAR, 5 Radar + AI Sensors

Example:Autonomous Vehicle

- **SAE 레벨 별 spec**



Sensor-L1	ASC*
Ultrasound	4
Radar(LRR)	1
Camera/ short-range LiDAR	1
SRR	2
Total	**6~8**

Sensor-L2	ASC
Ultrasound	8
Radar(LRR)	1
Radar(SRR)	2~4
Camera	2~4
Total	**~17**

Sensor-L3	ASC
Ultrasound	8
Radar(LRR)	2
Radar(SRR)	4
Camera(LR)	2
Camera(Sur)	4
Camera(Stereo)	1
µbolo*	1
LiDAR	1
Dead reckoning	1
Total	**24~26**

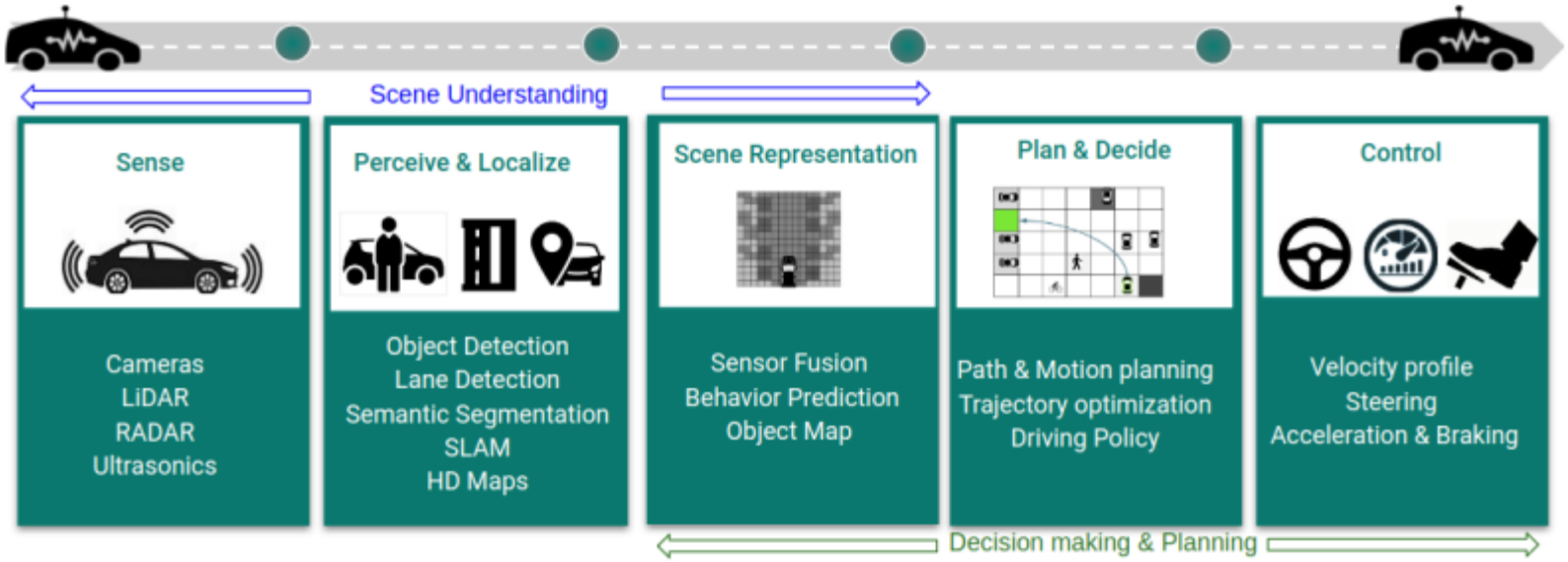
Sensor-L4	ASC
Ultrasound	8
Radar(LRR)	2
Radar(SRR)	4
Camera (stereo/trifocal)	2/3
Camera(Sur)	4
Camera(Stereo)	1
µbolo	1
LiDAR	2/4
Dead reckoning	1
Total	**25~28**

Sensor-L5	ASC
Ultrasound	8~10
Radar(LRR)	2
Radar(SRR)	4
Camera(LR)	2/3
Camera(Sur)	4
Camera(Stereo)	2
µbolo	1/2
LiDAR	4
Dead reckoning	1
Total	**28~32**

Example:Autonomous Vehicle

- **RL in Autonomous Vehicle**

- Scene Understanding: 환경을 정의하기 위해 센서 데이터 처리



Example:Autonomous Vehicle

- **RL in Autonomous Vehicle**

- 환경 모델링(State Space): Sensor data를 mid-level representation을 통해 환경 모델로 변환
  - Drivable Zone 탐색
  - 차선 정보 및 차선 변경 가능 여부
  - 교통 신호 및 보행자 정보
  - 주변 차량 및 장애물 감지(Occupancy Grid)
  - 시간-공간 정보(Time-to-collision, 과거 및 미래 궤적 예측)
  - Vehicle State Observation(lateral Offset, velocity, Yaw Angle)

- 데이터를 2D Bird-Eye- View(BEV) 맵 방식으로 변환하여 RL의 입력 state로 사용

Example:Autonomous Vehicle

- **RL in Autonomous Vehicle**

- Reward Function 설계
  - 안정한 주행
    - 목적지까지 이동 거리 증가 -> 양의 보상
    - 속도 유지 -> 양의 보상
    - 도로 차선 유지 -> 양의 보상
    - 신호 준수 -> 양의 보상
    - 충돌, 보행자와 가까운 거리 -> 패널티
  - 속도와 인간과 닮은 운전 방식
    - 급가속, 금정거, 급회전 -> 패널티
    - 스무스한 운전 -> 양의 보상