Training Set과 Test Set
- 같은 모집단에서 독립적으로 추출
- 비독립Collective Classification
- 다른 모집단 Transfer Learning

Agent120, Food30, Ghost12, Face4
World State $120*2^{30}*12^{2}*4$
Path State 120
Eat Dot State $120*2^{30}$

Search Graph : State가 1번만 등장
Search Tree : 각 노드가 Plan 대응

DFS : 스택
- 완전함(트리가 유한한 경우에만)
- 최적 아님(좌측의 해답을 찾음)
- 시간 복잡도 $O(B^{M})$ 전체 탐색
- 공간 복잡도 O(BM) 형제 노드만
- B 형제 노드의 수, M 깊이
- 걸리는 시간 짧지만 최적 아님

BFS : 큐
- 완전함(해답은 유한한 곳에 있음)
- 최적임(간선 비용이 동등할 때만)
- 시간 복잡도 $O(B^{S})$
- 공간 복잡도 $O(B^{S})$
- B 형제 노드 수, S 얕은 해답 깊이
- 걸리는 시간 길지만 최적임

Iterative Deepening : 부분트리의 높이를 1씩 증가시키며 DFS
- 반복되는 탐색 영역이 있지만 전체적으로 효율적

UCS : 누적 비용 우선순위 큐
- 완전함(음수 간선이 없다면)
- 최적임(음수 간선이 없다면)
- 균등한 비용이라면 BFS와 동치
- 시공간 복잡도 $O(B^{C*/\varepsilon})$
- 최적 해답의 비용 C*, 최소 간선 비용 ε, 유효 깊이 C*/ε

휴리스틱
- 유클리드 거리 : $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$
- 맨하탄 거리 : $|x_1-x_2|+|y_1-y_2|$
- Relaxation : 제약조건을 무시함
  - 따라서 최단거리의 하한값

Admissible Heuristic : $0 \le h(n) \le h^*(n)$
- 실제 값보다 휴리스틱 함수의 값이 작거나 같은 휴리스틱
- 모든 h(n)=0이면 UCS와 동등함

휴리스틱 결합 : max 결합
- 여전히 Admissible

Dominance : 모든 n에 $h_a(n) \ge h_b(n)$
- 즉, h_a가 항상 더 좋은 휴리스틱임

Consistency
- $h(A)-h(B) \le \mathrm{cost}(A,B)$
- Consistent하면 Admissible함

Graph Search : 이미 방문한 노드를 다시 방문하지 않음
- Consistency를 만족해야 함

그리디 알고리즘 : 휴리스틱 기반
- 최적 아님
- 안 완전함(DFS처럼 작동 가능)
- 백트래킹 없음 : 고려하는 선택지와 실제 선택지가 일치

A* 알고리즘 : g(n)+h(n)
- 최적임(Tree Search, Admissible)
- 최적(Graph Search, Consistency)
- 큐에서 노드를 꺼낼 때 종결

1. 최적해 A가 그렇지 않은 해 B보다 먼저 나오는지 증명
2. A가 B보다 먼저 큐에 들어가면 먼저 나오는 것은 당연함
3. B가 먼저 큐에 들어갔을 때는 다음과 같음
4. A의 조상 노드 n에 대해
$f(n)=g(n)+h(n) \le g(n)+h^*(n)=f(A$
, 즉 $f(n) \le f(A) \le f(B)$

Zerosum game : 각각의 에이전트가 서로 반대의 유틸리티를 가짐
- 보상의 합이 항상 0

General Game : 각각의 에이전트가 각각의 유틸리티를 가짐

Value of a State : 해당 State에서 얻을 수 있는 최고의 점수

Minimax : 적대적 에이전트와의 경쟁

```
def max_value(state):
  v = -INF
  for suc in state:
   v = max(v, value(suc))
  return v
```

```
def min_value(state):
  v = INF
  for suc in state:
   v = min(v, value(suc))
  return v
```

- DFS, 특히 Iterative Deepening
- DFS와 동일한 시공간 복잡도

Expectimax : 무작위 에이전트 상대
- 무작위거나 예측 불가인 상대, 실패 가능한 행동에 대하여 사용
- Depth-Limited Search 가능
- Minimax는 Expectimax의 부분집합

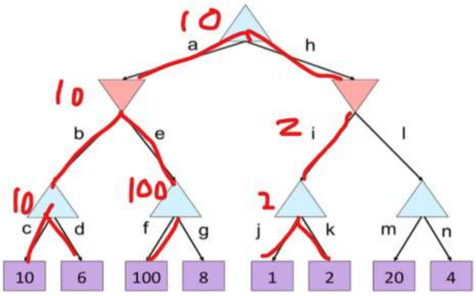
```
def exp_value(state):
  v = 0
  for suc in state:
   v += suc.prob * suc.val
  return v
```

Mixed Layer : 모든 Agent가 등장
Non Zerosum : 유틸리티가 튜플

Depth-Limited Search
- 리프 노드까지 탐색하지 못하므로 일부분까지만 탐색

Evaluation Function : 유틸리티의 추정치를 구하는 함수
- 최적해를 구하지 못함(불완전함)
- Anytime : 중간에 중단되어도 유효한 답을 낼 수 있음
- 가중치가 있는 함수들의 선형 결합

Alpha-Beta Prunning



- 중간 노드의 값에 영향을 미침
- 최종 노드에는 영향을 미치지 않음
- 최적의 경우 시간 복잡도 $O(B^{M})$에서 $O(B^{M/2})$로 떨어짐

```
init value of alpha = -INF
init value of beta = INF
```

```
def max_value(state, a, b):
  v = -INF
  for suc in state:
   v = max(v, value(suc, a, b))
   if v >= b: return v
   a = max(a, v)
  return v
```

```
def min_value(state, a, b):
  v = INF
  for suc in state:
   v = max(v, value(suc, a, b))
   if v <= a: return v
   b = min(b, v)
  return v
```

Monotonic Transformation
- Minimax Agent는 Insensitive
- Expectimax Agent는 Sensitive

	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

유틸리티

$A > B$: A가 B보다 낫다

$A \sim B$: 선호도가 같다

합리적 유틸리티의 원칙

Orderability

$(A > B) \vee (B > A) \vee (A \sim B)$

Transitivity

$(A > B) \wedge (B > C) \Rightarrow A > C$

Continuity

$A > B > C \Rightarrow \exists p\, [p, A; 1-p, C] \sim B$

- B와 선호도가 같은 지점을 A와 C의 조합으로 만들 수 있다
- pU(A) + (1-p)U(C) = U(B)

Substitutability

$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$

Monotonicity

$A > B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \succsim [q, A; 1-q, B])$

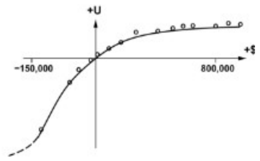
- A가 B보다 나을 때, p가 q 이상이라는 것은

pU(A)+(1-p)U(B)가 qU(A)+(1-q)U(B)라는 것과 동치이다

선형 변환에 대해, 행동은 변하지 않는다

- U\`(x) = k1U(x) + k2 (k1>0)

돈에 대해서, 유틸리티 곡선은 아래와 같은 경향이 있음



Stochastic World

Noisy Movement : 일정 확률로 결정한 방향이 아닌 쪽으로 움직임

MDP

- 상태의 집합 S
- 행동의 집합 A
- 전이함수 T(s, a, s\`) : 상태 s에서 행동 a를 해 s\`으로 이동 가능성
- 보상함수 R(s, a, s\`) : 상태 s에서 행동 a를 해 s\`으로 이동했을 때 보상

Marcov : 현재 상태에서, 다음 상태가 과거와 독립인 것(First Order)

Policy : 모든 상태에 대해 행동을 매핑하는 함수 $\pi : S \rightarrow A$

최적의 정책 : $\pi^* : S \rightarrow A$

Living Cost 적을 때 : 패배 상태에 빠지는 확률 최소화

Living Cost 높을 때 : 승리 상태에 빨리 도달하는 것이 중요

Living Cost가 극도로 높을 때 : 오래 사는 것보다 패배하는 게 이득

Decay Exponentially : 1, γ, γ^2,

- $0 \leq \gamma \leq 1$

Stationary Preference : $a > b$면 앞에 하나를 추가해도 $a > b$

- Additive / Discounted Utility만

무한한 게임

- Finite Horizon : 일정 깊이에 도달 시 게임 종결
- Discounting : 무한등비급수에 의해 수렴
- Absorbing State : 언젠가는 종료 상태에 도달

기대 유틸리티

V*(s) : s에서 시작해서 최적으로 행동 시의 기댓값

Q*(s,a) : s에서 a로 행동하는 걸로 시작해서 최적 행동 시 기댓값

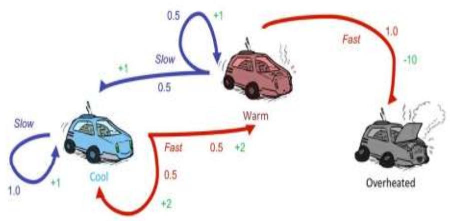
π*(s) : s에서 최적의 행동

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

- 액션의 확률과 (그때의 리워드와 이후의 V-function의 합)의 곱
- γ<1이면 유틸리티는 결국 수렴
- k=0일 때 모든 상태를 0으로
- 항상 고유한 최적값으로 수렴함
- 최적 정책은 최적값보다 빨리 수렴
- 반복마다의 복잡도는 $O(S^2A)$



V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

V_k와 V_{k+1}최대 γ^k max|R|만큼 차이나므로 수렴함