Week 0

Sunday, June 14, 2020 11:53 PM

Uninformed Search :
Search problem that use no problem-specific Knowledge

Informed Search :
Use problem-specific Knowledge

Uninformed Search
Depth-Firs-Search-->uses a Stack-->doesn't always find the best solution
Breadth-First-Search-->uses Queue-->always find the best solution

Algorithm for search problem :

1- find the state and the action
2- make the start node and add it to the frontier
3- define the explored set()
4- loop until you find the solution
5-inside the loop :
- A- if the frontier is empty then there is no solution
- B- remove node from the frontier
- C- add this node to the explored set
- D- find the neighbors(action and state) of this node
- E- loop over this neighbors
- F- for each neighbor check :
  - -is the state already not in the frontier
  - -is the in the state not in the explored set
- G- check if this neighbor is the target:
  - -if he is return the solution
  - -else add him to the frontier

**the solution:
-Loop until the node parent is none

1	Action = node.action
2	State = node.state
3	Node = node.parent
4	Append action and state to the end of the solution

-reverse the solution and return it

Informed Search
Greedy best-first Search:
Expand the node that is closest to the goal As estimated by heuristic function h(n)
Ex:: heuristic function --> Manhattan distance
May not found the best solution

A* Search :
Expand the node with the lowest value of g(n) + h(n)
-->g(n) = cost to reach the node
-->h(n) = estimated cost to reach the goal

A* found the best solution(optimal) if:
- -h(n) is admissible (never overestimates the true cost)
- -h(n) is consist
  - (for every node n and successor n` with step cost c : h(n) <= h(n`) + c)

Adversarial Search
Uses MinMax:
- Max: aim to maximize the score
- Min : aim to minimize the score

Ex: game like tik tak tok

Approach :
->S_0 : initial state
->Player(s): return which player to move in a state S
->Actions(s): return legal moves in the state S
->Result(s, a): return state after an action a is taken
->Terminal(s): return if a state is a terminal state (end the game)
->Utility(s): final numerical value for terminal state S(who won ??)

MinMax
Given a state S :
- -Max pick an action a in Actions(s) that produce the height value of the Min-Value(Result(s, a))
- -Min pick an action a in Actions(s) that produce the lowest value of the Max-Value(Result(s, a))

Function Max-Value(S):
if the game is over return the result
If Terminal(s):

Function Min-Value(S):
if the game is over return the result
If Terminal(s):

Optimization

Alpha-Beta pruning:
Keeping tack of the best and the worst I can do so far

Depth-Limited MinMax :
Stop after limited number of moves and use an evaluation function
Evaluation function:
Estimate the expected utility result of the game from a given state

```
Function Max-Value(S):
## if the game is over return  the
result
        If Terminal(s):
                Return utility(s)
        v = -infinity
        For action in Actions(S):
                v = Max(v, Min-
                Value(Result(S,
                action)))
        Return v
```

```
Function Min-Value(S):
## if the game is over return  the
result
        If Terminal(s):
                Return utility(s)
        v = infinity
        For action in Actions(S):
                v = Min(v, Max-
                Value(Result(S, action)))
        Return v
```