Honey Bee

1.1 BUILDING YOUR WORLD

Peas Description :

Performance :

- Honey Bee move forward = -10
- Honey Bee find the Cactus = -20
- Honey Bee find the flower with nectar = +50
- Honey Bee find the flower with nectar = +100

Environment :

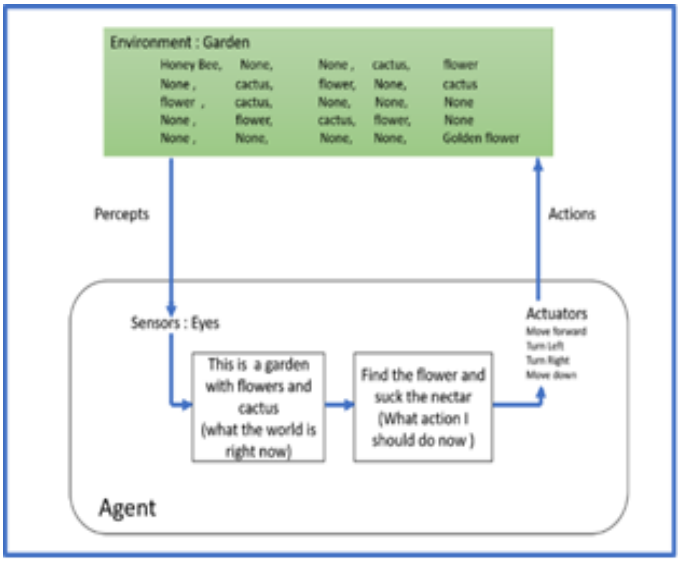
- Garden with flowers and Cactus

Actuators :

- Move forward
- Turn right
- Turn left
- Move down

Sensor :

- Find the flower
- Smell the nectar



Simple Reflex Agent

Simple Reflex Agent

These agents select actions on the basis of the current percept, ignoring the rest of the percept history.
Here in my 2d game the honey bee flies as per the given iteration (here we have given 20) he did not look for previous history and try to find the flower and suck the nectar and if he find cactus then it die and gave over.

```
************************ Reflex Agent ****************************************
HoneyBee flied to  (1, 0)
Nothing present in the location
HoneyBee flied to  (2, 0)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (3, 0)
Nothing present in the location
HoneyBee flied to  (2, 0)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (2, 1)
HoneyBee Died : due to  cactus
Total steps performed :  5
Performance :  60
******************************************************************************
```

Advantages :

- These agents simply decide actions based on the their current percept.

(here honey bee is flying based on the given steps).

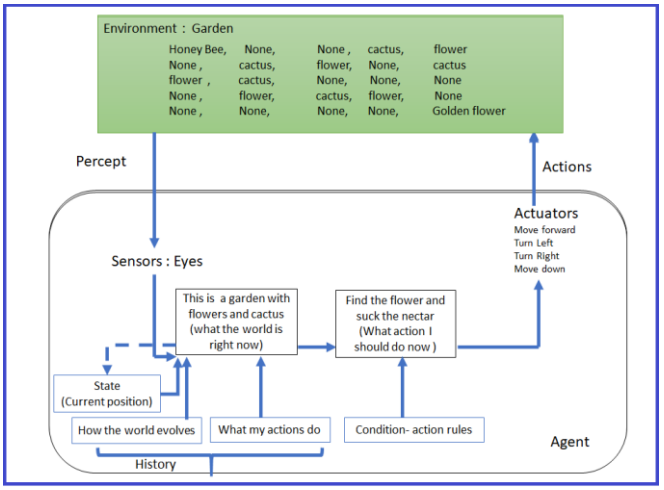
Disadvantages :

- Very limited intelligence.
  (The honey bee does not have any knowledge of previous path to find the flower)
- No knowledge of non-perceptual parts of state.
  (honey bee does not have any clues where is the flowers and cactus)

Model Based Reflex Agent

The agent has to keep track of **internal state** which is adjusted by each percept and that depends on the percept history.

In my 2d honeybee game it keeps track of percept and it's action
("flower", "suck"),("cactus", "Die")



Model Based Reflex Agent

Advantages :

- It stores the percept history.
  (here we are storing previous percept history)
- Runs faster because it has previous history data.

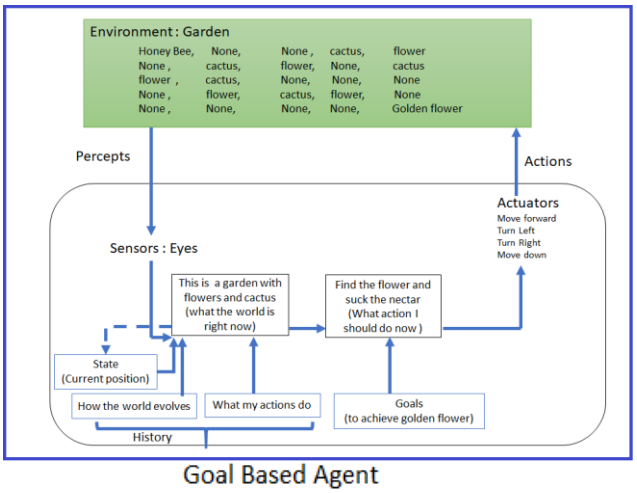
Disadvantages :

- Space complexity (It stores the data)

```
HoneyBee flied to  (2, 4)
Nothing present in the location
HoneyBee flied to  (2, 4)
Nothing present in the location
HoneyBee flied to  (2, 3)
Nothing present in the location
HoneyBee flied to  (3, 3)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (4, 3)
Nothing present in the location
HoneyBee flied to  (3, 3)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (3, 4)
Nothing present in the location
maintain percept history :  {'cactus': 'Die', 'flower': 'suck'}
**********************************************************************
```

Goal Based Agent :

This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.
Here in my game the goal of honey bee is to reach the golden flower and suck the nectar. The performance is calculating in each step (flied forward and nothing in that position then -10, when it finds cactus then -20, when it finds flower then +50 and when it reaches to goal then +100).



Goal Based Agent

Advantages :

- It stores the percept history.(here we are storing previous percept history)

Disadvantages :

- It only reaches to the goal but it does not consider optimal path to reach the goal

```
HoneyBee flied to  (3, 2)
Nothing present in the location
HoneyBee flied to  (3, 3)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (3, 4)
Nothing present in the location
HoneyBee flied to  (2, 4)
Nothing present in the location
HoneyBee flied to  (2, 3)
Nothing present in the location
HoneyBee flied to  (3, 3)
HoneyBee suck the nectar from flower :  suck
HoneyBee flied to  (3, 4)
Nothing present in the location
HoneyBee suck the nectar from goldenflower at ( 5 ,  5 ) : Goal Achived
Total step performed :  152
Performance :  200
**************************************************************************************
```

Conclusion :

- 2-diamential world
    - Easy to read and interpret
    - Easy to change
    - Easy to distribute and multiply.
- Simple Reflex Agent : Honey Bee flies till it completes the iteration or it dies if he find the cactus.
- Model Based Agent : Honey Bee stores the percept and action into its memory.
- Goal Based Agent : Honey Bee files until it finds the goal(Golden Flower) and stores percept history.

1.2 SEARCHING YOUR WORLD

Problem statement :

- **States**: The state is determined by Honey Bee location and the Golden Flower locations. The agent is in [0,0] location and Golden Flower is at [4,4].
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: In this simple environment, the Honey Bee flies right , left, up, down
- **Transition model**: The Honey bee moves from one location to another and try to suck the nectar from the flower. If it finds cactus it dies ,the honey bee moves from left to right and up and down
- **Goal test**: To reach the Golden Flower(Goal State) and try to suck the nectar from it.
- **Path cost**: each steps has different cost.

Search : It is a mechanism to follow the sequence of steps to reach the goal but we do not know the actual path to reach the goal, so it a systematic trial and error method to determine the actual minimal path.

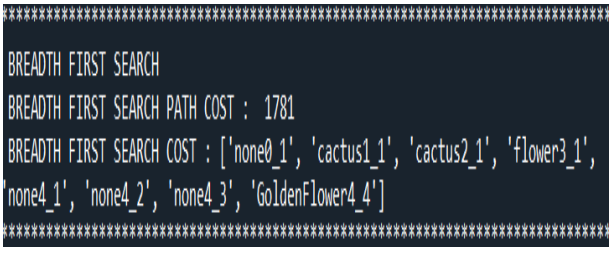
Garden_map.locations = dict(
HoneyBee=(0, 0), none0_1=(0, 1),none0_2=(0, 2), cactus0_3=(0, 3),flower0_4=(0, 4),
none1_0=(1, 0), cactus1_1=(1, 1), flower1_2=(1, 2),none1_3 = (1,3),cactus1_4=(1,4),
flower2_0=(2, 0), cactus2_1=(2, 1),none2_2=(2,2), none2_3=(2, 3),none2_4=(2,4),
none3_0=(3, 0), flower3_1=(3, 1), cactus3_2=(3, 2),flower3_3 =(3,3),none3_4 =(3,4),
none4_0=(4, 0), none4_1=(4, 1), none4_2=(4, 2),none4_3= (4,3),GoldenFlower4_4 =(4,4))

Uninformed Search : It does not have additional information about the distance from Honey Bee location (initial state) to Golden Flower(Goal state).

- **Initial state** : Honey Bee=(0, 0) **Goal state** : GoldenFlower4_4 =(4,4)

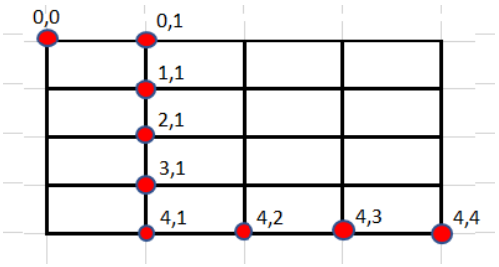
1. Breadth First Search : .

- First visit all horizontal nodes and then it visits each level.
- Here the algorithm starts [0,0] and then it selects none0_1 and then selects the horizontal locations {cactus1_1=(1, 1), cactus2_1=(2, 1), flower3_1=(3, 1),



none4_1=(4, 1)}and then its go vertically { none4_2=(4, 2), none4_3= (4,3), GoldenFlower4_4 =(4,4)}

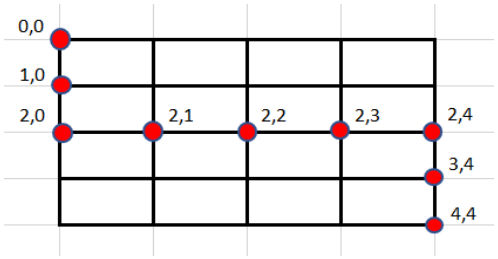
Path Cost : 1781



2. DEPTH-FIRST GRAPH SEARCH :

- Uses the stack to remember to select the next node to start a search.
- Select the adjacent unvisited node (none1_0). Mark that node as selected and display that node. Push that node into stack
- If he did not find the adjacent node is found, pop up a node from the stack
- And that repeat the above steps
- **Path Cost : 1056**

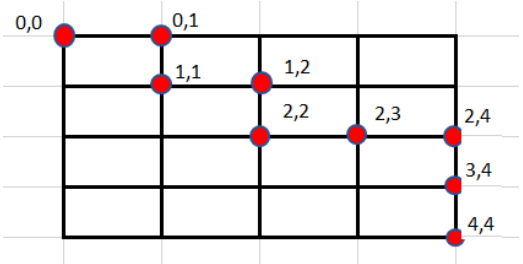
```
**********************************************************************************************************
DEPTH-FIRST GRAPH SEARCH
DEPTH-FIRST GRAPH SEARCH PATH COST :  1056
DEPTH-FIRST GRAPH SEARCH PATH :  ['none1_0', 'flower2_0', 'cactus2_1', 'none2_2',
'none2_3', 'none2_4', 'none3_4', 'GoldenFlower4_4']
**********************************************************************************************************
```



3. UNIFORM COST SEARCH :

- Find the path which has lowest cumulative cost.
- It expands nodes according to their path costs from the start node
- This is optimal because it select lowest cost path.
- Here honeybee selected lowest cost nodes to reach the golden flower.
- **Path Cost : 917**

```
**********************************************************************************************************
UNIFORM COST SEARCH
UNIFORM COST SEARCH PATH COST :  917
UNIFORM COST SEARCH PATH :  ['none0_1', 'cactus1_1', 'flower1_2', 'none2_2', 'none2_3',
'none2_4', 'none3_4', 'GoldenFlower4_4']
**********************************************************************************************************
```

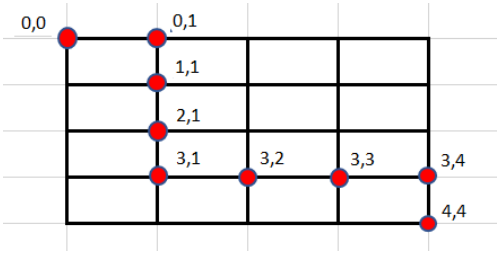


Informed Search : It has some additional information about the evaluation of the distance from Honey Bee location (initial state) to Golden Flower(Goal state).

4. Best First Search :

- This uses the concept of heuristic approach
- It calculates the cost of adjacent node pf honey bee and select which is minimal path and select the node which has minimum cost.
- **Path Cost : 917**

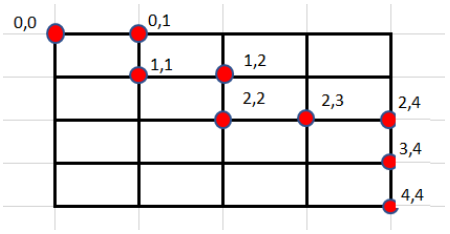
```
**********************************************************************
BEST FIRST SEARCH
BEST FIRST SEARCH PATH COST :  1206
BEST FIRST SEARCH PATH :  ['none0_1', 'cactus1_1', 'cactus2_1', 'flower3_1',
'cactus3_2', 'flower3_3', 'none3_4', 'GoldenFlower4_4']
**********************************************************************
```



5. A* Search :

- This uses the formula **f(n) = g(n) + h(n)**
- $f(n)$ = total estimated cost of path from Honeybee location
- g(n) = cost so far to reach golden flower
- h(n) = estimated cost from Honeybee location to Golden flower. (This includes heuristic part)
- **Path Cost : 917**

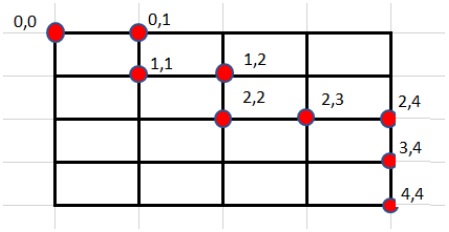
```
**********************************************************************
A* SEARCH
A* SEARCH PATH COST :  917
A* SEARCH PATH :  ['none0_1', 'cactus1_1', 'flower1_2', 'none2_2', 'none2_3',
'none2_4', 'none3_4', 'GoldenFlower4_4']
**********************************************************************
```



6. RECURSIVE BEST FIRST SEARCH :

- that is linear with respect to the maximum search depth, regardless of the cost function used
- It works by maintaining on the recursion stack the complete path to the current node(honey bee location) being expanded as well as all immediate connected of nodes on that path, along with the cost of the best node in the sub-tree explored below each sibling.
- **Path Cost : 917**

```
**********************************************************************
RECURSIVE BEST FIRST SEARCH
RECURSIVE BEST FIRST SEARCH PATH COST :  917
RECURSIVE BEST FIRST SEARCH PATH :  ['none0_1', 'cactus1_1', 'flower1_2', 'none2_2',
'none2_3', 'none2_4', 'none3_4', 'GoldenFlower4_4']
**********************************************************************
```



1.3 FORWARD-CHAINING AND BACKWARD-CHAINING

Inference Engine : It applies rules(logical) to the knowledge to conclude new information form the already known facts.

Following are the two nodes :

- **Forward Chaining :**
  - Bottom-Up approach
  - It reaches to goal and uses some already known facts.
  - It uses available data(data-driven).

- **based on the rules and available data we start from initial location to the goal.**

Step 1: We start from which we already have information about the facts, as per the facts we choose the next sequential steps.
Example :

Available Data : `
[0,0] --->[1,0]
[1,0] --->[1,1]
[0,0] --->[1,1]

Rules : If [0,0] ---> [1,1] is True then [4,4] is True

Step 2: Now, we use those facts and conclude the satisfied conditions.
Now, Check all conditions which satisfied the location [0,0] -->[1,1] and connect the next node which satisfied the given rules

Step 3:in this step we need to check the given statement and check if it is satisfied with the previous stated facts so that you can reach the goal

Example :
1.Honey Bee flies from initial location to the Golden flower(goal).
2.If Honey bee reaches to the golden flower then it sucks the nectar.

Conclusion :
Honey Bee suck the nectar from the Golden flower.

- **Backward Chaining :**
  - Top-Down approach
  - It divides into sub-goals and it prove the facts true.
  - They have list of goals and that decides which one should selected.

Step 1. First step is to check the goal facts and will obtain other facts that should be True.

Available Data :

[1,1] --->[1,0]

[1,0] --->[0,0]

[1,1] --->[0,0]

Rules : If [1,1] --->[0,0] is True then [4,4] is True

Based on the rules and available goal facts, we start process from initial location and move towards goal state.

Step 2: Obtain other facts from the goal facts and that should be satisfy with the rules.

Step 3: extract further fact which infers from facts inferred from the step 2

Step 4: repeat the same until we get to a certain fact that satisfies the conditions.

Example :

1.Honey Bee suck the nectar from the Golden flower.

2.Honey bee reaches to the golden flower(goal) from start location.

Conclusion :

Honey Bee flies from initial location to the Golden flower(goal).

Forward Chaining	Backward Chaining
A bottom-up approach	A top-down approach
Data-driven technique	Goal-driven technique
Slow as it has to use all the rules	Fast as it has to use only a few rules
It works from initial state to goal state(forward direction	It works from goal state to initial state(backward direction)

Reference : aima Repository, Artificial Intelligence A Modern Approach(Third Edition)