KNOWLEDGE REPRESENTATION

ASSIGNMENT – 1 REPORT

TASK – 1: BUILDING YOUR WORLD

2D WORLD:

The environment which is being considered over here is a "Garden" which holds different agents like 'Squirrel', 'Rat', and 'Dog' and objects(things) like 'Nuts', 'Carrot', 'Home', 'Owner', 'Ball' and 'Obstacles'. The Garden is a 2-Dimensional world in which each block's location is identified by (x, y) coordinates as in a 2D plane. Each block in the environment holds an object or thing, which serves as a playable or source of energy for the agents. Each agent in the environment has their tasks to perform through which they achieve some score or points which can be used to analyze the performance of an agent. The agents which are being considered over here are "SIMPLE REFLEX AGENT", "MODEL-BASED REFLEX AGENT" and "GOAL BASED AGENT".

CUTE SQUIRREL AGENT – SIMPLE REFLEX AGENT(SRA):

The Cute Squirrel Agent is a Simple Reflex Agent which performs its actions by randomly wandering the environment. The Squirrel can move in four directions i.e., 'left', 'right', 'up', 'down'. The agent will have no prior knowledge of the environment and doesn't store any information about the environment. The task environment of this agent is described as bellow

PEAS Description

Agent Type	**Performance Measure**	**Environment**	**Actuators**	**Sensors**
Cute Squirrel Agent	Eating Nuts, throwing bad nuts, avoiding other things	Garden or Park	Hands, Legs, and Mouth	Eyes and Ears

The Squirrel's task is to move randomly through the garden, eat good nuts and throw bad nuts which results in a gain of 10 points and a loss of 1 point respectively, and avoids anything apart from a nut. The agent will stop moving if completes eating all the nuts. As the Cute Squirrel is a 'simple reflex agent' it will select the actions based on the current percept and ignore the percept history.

- **Advantages and Disadvantages:**
    - Performs the actions based on the current state of the world and rules.
    - Very Simple and sometimes robust.
    - No memory and a Limited range of applicability.
- With no memory and a limited range of applicability, this agent has less performance in a 2D World.
- The agent takes more time and steps to complete his task with the increase of the size of the world.

SNEAKY RAT – MODEL-BASED REFLEX AGENT(MBRA):

The Sneaky Rat Agent is a Model-Based Reflex Agent which performs its actions with the help of percept history which is stored as an internal state by the agent. The Rat can move in four directions i.e., ‘left’, ‘right’, ‘up’, ‘down’, avoiding the obstacles in its paths based on his previous percept history of the environment. The task environment of this agent is described as bellow

PEAS Description

Agent Type	**Performance Measure**	**Environment**	**Actuators**	**Sensors**
Sneaky Rat Agent	Eating carrots, avoiding obstacles	Garden or Park	Hands, Legs, and Mouth	Eyes and Ears

The task of the rat agent is to reach home, which can be achieved by making decisions on the next step based on the previous percept history and current state of the environment.

- **Advantages and Disadvantages:**
  - Reasons with past state of the world and acts based on the current state of world and rules.
  - Stores the previous state, but still has a limited range of applicability.
- The agent performs better than SRA but still has limitations in achieving the task.
- The agent takes more time and steps to complete his task with the increase of the size of the world but performs better compared to SRA.

PET DOG – GOAL-BASED AGENT(GBA):

The Pet Dog is a Goal-Based Agent which makes the smart choice to move in the environment in a way that each step it takes will make the agent close to its goal. The agent stores the percept history and makes some decisions that will help to reach its goal. The task environment of this agent is described as bellow

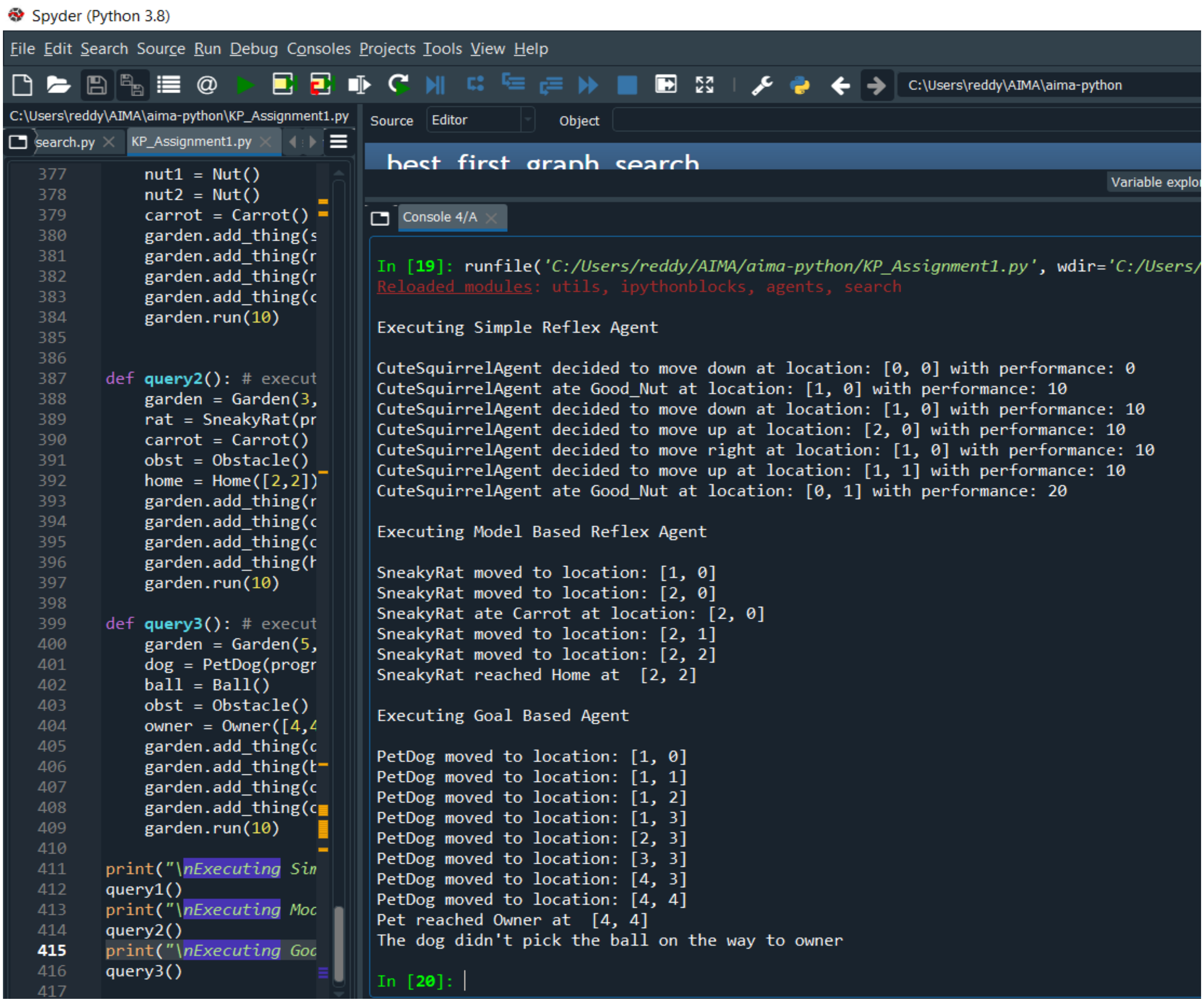
PEAS Description

Agent Type	**Performance Measure**	**Environment**	**Actuators**	**Sensors**
Pet Dog Agent	Picking ball, avoiding obstacles	Garden or Park	Hands, Legs, and Mouth	Eyes and Ears

The dog’s goal is to reach its owner and, on the way, to pick the ball. We use the distance between two points to choose the next step which makes the dog closer to its owner by each step.

- **Advantages and Disadvantages:**

   - Able to reason with initial, goal, and intermediate.
   - Can’t feasibly solve many general problems.
- The agent achieves its task with limited steps.
- The size of the environment doesn’t matter as the agent finds takes each step towards his goal.
- Output:


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\reddy\AIMA\aima-python
C:\Users\reddy\AIMA\aima-python\KP_Assignment1.py
search.py
KP_Assignment1.py
Source
Editor
Object
best first graph search
Variable explor
Console 4/A

```
    nut1 = Nut()
    nut2 = Nut()
    carrot = Carrot()
    garden.add_thing(s
    garden.add_thing(r
    garden.add_thing(r
    garden.add_thing(c
    garden.run(10)


def query2(): # execut
    garden = Garden(3,
    rat = SneakyRat(pr
    carrot = Carrot()
    obst = Obstacle()
    home = Home([2,2])
    garden.add_thing(r
    garden.add_thing(c
    garden.add_thing(c
    garden.add_thing(h
    garden.run(10)

def query3(): # execut
    garden = Garden(5,
    dog = PetDog(progr
    ball = Ball()
    obst = Obstacle()
    owner = Owner([4,4
    garden.add_thing(c
    garden.add_thing(b
    garden.add_thing(c
    garden.add_thing(c
    garden.run(10)

print("\nExecuting Sim
query1()
print("\nExecuting Moc
query2()
print("\nExecuting Goc
query3()
```

```
In [19]: runfile('C:/Users/reddy/AIMA/aima-python/KP_Assignment1.py', wdir='C:/Users/
Reloaded modules: utils, ipythonblocks, agents, search

Executing Simple Reflex Agent

CuteSquirrelAgent decided to move down at location: [0, 0] with performance: 0
CuteSquirrelAgent ate Good_Nut at location: [1, 0] with performance: 10
CuteSquirrelAgent decided to move down at location: [1, 0] with performance: 10
CuteSquirrelAgent decided to move up at location: [2, 0] with performance: 10
CuteSquirrelAgent decided to move right at location: [1, 0] with performance: 10
CuteSquirrelAgent decided to move up at location: [1, 1] with performance: 10
CuteSquirrelAgent ate Good_Nut at location: [0, 1] with performance: 20

Executing Model Based Reflex Agent

SneakyRat moved to location: [1, 0]
SneakyRat moved to location: [2, 0]
SneakyRat ate Carrot at location: [2, 0]
SneakyRat moved to location: [2, 1]
SneakyRat moved to location: [2, 2]
SneakyRat reached Home at  [2, 2]

Executing Goal Based Agent

PetDog moved to location: [1, 0]
PetDog moved to location: [1, 1]
PetDog moved to location: [1, 2]
PetDog moved to location: [1, 3]
PetDog moved to location: [2, 3]
PetDog moved to location: [3, 3]
PetDog moved to location: [4, 3]
PetDog moved to location: [4, 4]
Pet reached Owner at  [4, 4]
The dog didn't pick the ball on the way to owner

In [20]:
```

TASK – 2: SEARCHING YOUR WORLD

Problem Statement:

The tourists in Paris have got a problem with finding the routes from one place to another in the city. So, we are going to solve their problem using the Graph Problem. Collecting the information regarding all the famous places in Paris and preparing a graph as such each place is connected with others and has the distance as cost. The main problem here is to find the possible route and their cost for “Eiffel Tower to Royal Palace”, “Holy Chapel to River Cruise”, and “Rodin Museum to Alexander Bridge”. We are going to apply different search algorithms to them and compare those results to find the best way to travel from one place to another with less cost.

Goal State:

The goal stated in this problem is to reach “Royal Palace”, “River Cruise”, and “Alexander Bridge” from their respective initial positions. To achieve this we are going to use three uninformed search techniques and three informed search techniques.

The program uses “Graph Problem” from the search.py file from the AIMA repository, which helps us to achieve our goal. We are going to compare all the searches and analyze their performance with the problem.

Uninformed Search Techniques:

- Breadth-First Graph Search: The Breadth-First Graph Search starts from a root node and explores its neighbors first and then it moves to the next level. Which helps us to search nearby places first which helps us to minimize the movement in the search. In our problem, this search technique takes fewer actions compared to others.

```
Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

< 12/ 14/  28/Roya>            <   2/   4/   6/Rive>          <   7/  11/  18/Alex>
```

- Depth First Graph Search: This search will traverse through the graph from the root node and expand to its deepest node. It improves memory by deleting all the nodes it has traversed. It usually takes less time than BFS, but in our problem, it is taking more time than BFS as the goal states of our problem are within the breadth of the graph.

```
Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

    < 10/ 11/ 24/Roya>          <   8/   9/ 19/Rive>          <  10/  11/  24/Alex>
```

- Iterative Deepening Search: This search technique will help us to identify the best depth limit, by increasing the limit until the goal is achieved. It increases the runtime for exploring all the depths. It takes more runs to move from Eiffel Tower to Royal Palace while the rest are normal.

```
Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

    <  54/ 159/ 157/Roya>         <   3/   8/   7/Rive>          <  10/  29/  29/Alex>
```

Informed Search Techniques:

- Best First Graph Search: The Best First Graph Search selects the best path at that moment. It is a combination of both BFS and DFS and switches between which gives an advantage of finding the goal quickly.

Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

< 11/ 12/ 27/Roya> < 7/ 8/ 18/Rive> < 9/ 10/ 23/Alex>

- Uniform Cost Search: Uniform Cost Search is the best algorithm for search and does not use heuristics. It gives maximum priority to the lowest cost and is equivalent to BFS. Even it shows better results than other informed search techniques.

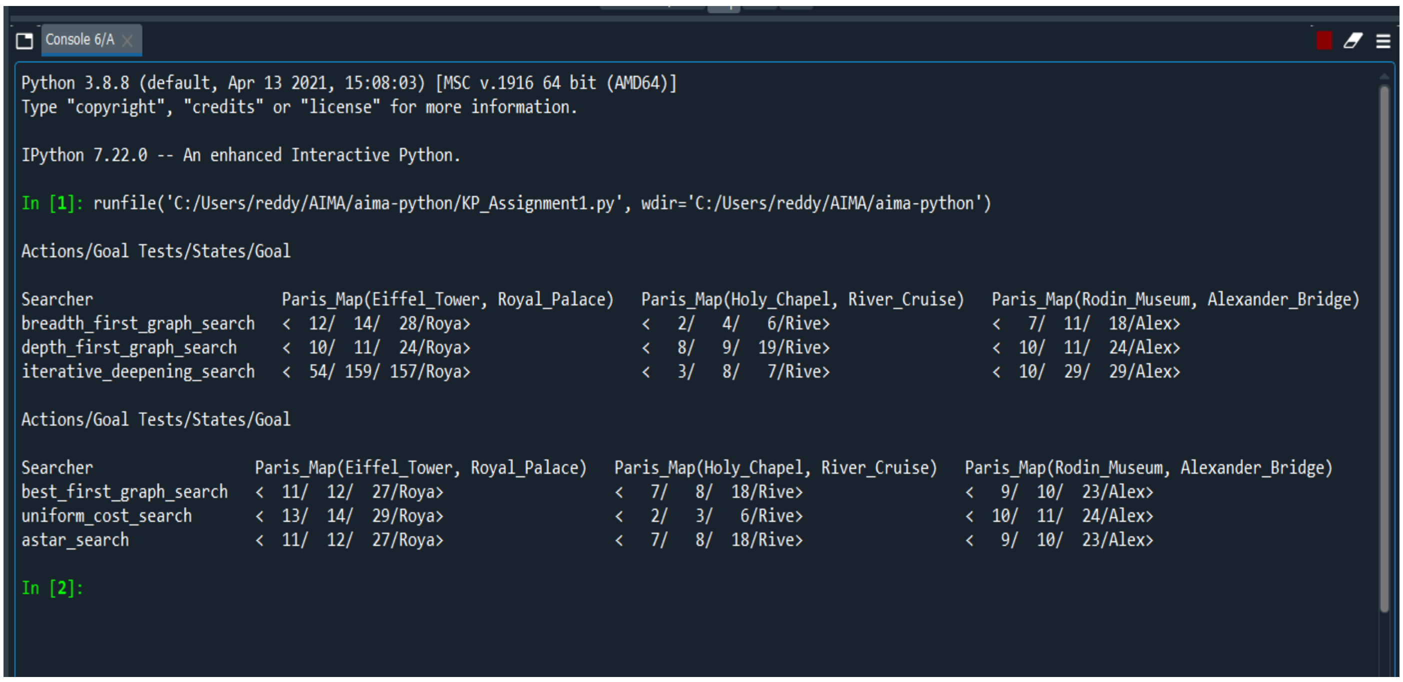
Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

< 13/ 14/ 29/Roya> < 2/ 3/ 6/Rive> < 10/ 11/ 24/Alex>

- A* Graph Search: This search technique will demand more consistency from the heuristic, which helps in producing a more optimal solution. I have tried this search without giving any heuristic function and that resulted in me with the same result as the best-first search algorithm.

Eiffel Tower to Royal Palace| |Holy Chapel to River Cruise| |Rodin Museum to Alexander Bridge

< 11/ 12/ 27/Roya> < 7/ 8/ 18/Rive> < 9/ 10/ 23/Alex>



```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/reddy/AIMA/aima-python/KP_Assignment1.py', wdir='C:/Users/reddy/AIMA/aima-python')

Actions/Goal Tests/States/Goal

Searcher                     Paris_Map(Eiffel_Tower, Royal_Palace)   Paris_Map(Holy_Chapel, River_Cruise)   Paris_Map(Rodin_Museum, Alexander_Bridge)
breadth_first_graph_search   <  12/  14/  28/Roya>                   <   2/   4/   6/Rive>                  <   7/  11/  18/Alex>
depth_first_graph_search     <  10/  11/  24/Roya>                   <   8/   9/  19/Rive>                  <  10/  11/  24/Alex>
iterative_deepening_search   <  54/ 159/ 157/Roya>                   <   3/   8/   7/Rive>                  <  10/  29/  29/Alex>

Actions/Goal Tests/States/Goal

Searcher                  Paris_Map(Eiffel_Tower, Royal_Palace)   Paris_Map(Holy_Chapel, River_Cruise)   Paris_Map(Rodin_Museum, Alexander_Bridge)
best_first_graph_search   <  11/  12/  27/Roya>                   <   7/   8/  18/Rive>                  <   9/  10/  23/Alex>
uniform_cost_search       <  13/  14/  29/Roya>                   <   2/   3/   6/Rive>                  <  10/  11/  24/Alex>
astar_search              <  11/  12/  27/Roya>                   <   7/   8/  18/Rive>                  <   9/  10/  23/Alex>

In [2]:
```