Experiment 07 – Knowledge Base Agent

Learning Objective: Student should be able to build knowledge base for Wumpus world problem.

Tools: Python under Windows or Linux environment

Theory: Study and implement Wumpus World Problem.

A variety of "worlds" are being used as examples for Knowledge Representation, Reasoning, and Planning. Among them the Vacuum World, the Block World, and the Wumpus World. We will examine the Wumpus World and in this context introduce the Situation Calculus, the Frame Problem, and a variety of axioms.

The Wumpus World was introduced by Genesereth, and is a simple world (as is the Block World) for which to represent knowledge and to reason.

It is a cave with a number of rooms, represented as a 4x4 square.

```
   +---------+---------+---------+---------+
 4 | stench  |         | breeze  |  pit    |
   |         |         |         |         |
   |         |         |         |         |
   +---------+---------+---------+---------+
 3 | wumpus  | stench  | pit     | breeze  |
   |         | breeze  |         |         |
   |         | gold    |         |         |
   +---------+---------+---------+---------+
 2 | stench  |         | breeze  |         |
   |         |         |         |         |
   |         |         |         |         |
   +---------+---------+---------+---------+
 1 | start   | breeze  | pit     | breeze  |
   |         |         |         |         |
   |   ==>   |         |         |         |
   +---------+---------+---------+---------+
        1         2         3         4
```

Rules of the Wumpus World

The **neighborhood** of a node consists of four squares north, south, east, west of the given square.

In a square the agent gets percepts, with components **Stench, Breeze, Glitter, Bump, Scream**

For example [Stench, None, Glitter, None, None]

- Stench is perceived at a square iff the wumpus is at this square or in its neighborhood.
- Breeze is perceived at a square iff a pit is in the neighborhood of this square.
- Glitter is perceived at a square iff gold is in this square
- Bump is perceived at a square iff the agent goes Forward into a wall
- Scream is perceived at a square iff the wumpus is killed anywhere in the cave

An agent can do the following actions (one at a time):

Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb

- The agent can go Forward in the direction it is currently facing, or Turn Right, or Turn Left. Going Forward into a wall will generate a Bump percept.
- The agent has a single arrow that it can Shoot. It will go straight in the direction faced by the agent until it hits (and kills) the wumpus, or hits (and is absorbed by) a wall.
- The agent can Grab a portable object at the current square or it can Release an object that it is holding.
- The agent can Climb out of the cave if at the Start square.

The Start square is (1,1) and initially the agent is facing east. The agent dies if it is in the same square as the wumpus.
The objective of the game is to kill the wumpus, to pick up the gold, and to climb out with it.

Representing our Knowledge about the Wumpus World

Percept(x,y) where x must be a percept vector and y must be a situation. It means that at situation y the agent perceives x.
For convenience we introduce the following definitions:

- Percept([Stench,y,z,w,v],t) = > Stench(t)
- Percept([x,Breeze,z,w,v],t) = > Breeze(t)
- Percept([x,y,Glitter,w,v],t) = > AtGold(t)

Holding(x,y) where x is an object and y is a situation. It means that the agent is holding the object x in situation y.

Action(x,y) where x must be an action (i.e. Turn(Right), Turn(Left), Forward, ..) and y must be a situation. It means that at situation y the agent takes action x.

At(x,y,z) where x is an object, y is a Location, i.e. a pair [u,v] with u and v in {1,2,3,4}, and z is a situation. It means that the agent x in situation z is at location y.

Present(x,s) means that object x is in the current room in the situation s.

Result(x,y) It means that the result of applying action x to the situation y is the situation Result(x,y). Note that Result(x,y) is a term, not a statement.

For example we can say

- Result(Forward, S0) = S1
- Result(Turn(Right),S1) = S2

<u>Inference Rules:</u>

We can prove that wumpus is in the room (1, 3) using propositional rules which we have derived for the wumpus world and using inference rule.

- Apply Modus Ponens with ¬S11 and R1:

We will firstly apply MP rule with R1 which is $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$, and $\neg S_{11}$ which will give the output $\neg W_{11} \wedge W_{12} \wedge W_{12}$.

- Apply And-Elimination Rule:

After applying And-elimination rule to $\neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$, we will get three statements:

$\neg W_{11}$, $\neg W_{12}$, and $\neg W_{21}$.

- Apply Modus Ponens to $\neg S_{21}$, and R2:

Now we will apply Modus Ponens to $\neg S_{21}$ and R2 which is $\neg S_{21} \rightarrow \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$, which

will give the Output as $\neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

- Apply And -Elimination rule:

  Now again apply And-elimination rule to $\neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$, We will get three statements: $\neg W_{21}$, $\neg W_{22}$, and $\neg W_{31}$

- Apply MP to S_{12} and R4:

  Apply Modus Ponens to S_{12} and R 4 which is $S_{12} \rightarrow W_{13} \vee. W_{12} \vee. W_{22} \vee. W_{11}$, we will get the output as $W_{13} \vee W_{12} \vee W_{22} \vee. W_{11}$.

- Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$:

  After applying Unit resolution formula on $W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$ and $\neg W_{11}$ we will get $W_{13} \vee W_{12} \vee W_{22}$.

- Apply Unit resolution on $W_{13} \vee W_{12} \vee W_{22}$ and $\neg W_{22}$:
  After applying Unit resolution on $W_{13} \vee W_{12} \vee W_{22}$, and $\neg W_{22}$, we will get $W_{13} \vee W_{12}$ as output.

- Apply Unit Resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$:
  After Applying Unit resolution on $W_{13} \vee W_{12}$ and $\neg W_{12}$, we will get W_{13} as an output, hence it is proved that the Wumpus is in the room [1, 3].

<u>Design:</u>

```
#9=stench

#8=glitter

#7=pit

#6=gold

#5=breeze

#-1=wumpus


def learnagent(world,i,j):

‘‘‘Function for an agent to know what poisitin contains which environment objects’’’

if (world[i][j]==9):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))

print(“You came across a stench”)

return agi,agj


elif (world[i][j]==8):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))

print(“You came across a glitter”)

return agi,agj

elif (world[i][j]==7):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))
```

```
print(“You came across a pit”)

return -5,-5

elif (world[i][j]==6):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))

print(“You found gold”)

return -4,-4

elif (world[i][j]==5):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))


print(“You feel breeze”)

return agi,agj

elif (world[i][j]==-1):

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))

print(“You met wumpus”)

return -5,-5

else: #if world environment was empty

agi,agj=i,j

print(“\nNow the agent is at “+str(agi)+”,”+str(agj))

return agi,agj
```

```
def checkinp(agi,agj):

‘‘‘Function for checking input going in forward direction to get gold’’’

if(agi==0 and agj==0):

print(“\nyou can go at “+str(agi+1)+” “+str(agj)) #can move upward

print(“you can go at “+str(agi)+” “+str(agj+1)) #can move right

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))


if(agvi==agi+1 and agvj==agj or agvi==agi and agvj==agj+1):

return agvi,agvj

else:

return -5

elif(agi==3 and agj==0):

print(“\nyou can go at “+str(agi-1)+” “+str(agj)) #can go left

print(“you can go at “+str(agi)+” “+str(agj+1)) #can go right

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi-1 and agvj==agj or agvi==agi and agvj==agj+1):

return agvi,agvj

else:

return -5

elif(agi==3 and agj==3):

print(“\nyou can go at “+str(agi-1)+” “+str(agj)) #can go down
```

```
print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))


if(agvi==agi-1 and agvj==agj or agvi==agi and agvj==agj-1):

return agvi,agvj

else:

return -5

elif(agi==0 and agj==3):

print(“\nyou can go at “+str(agi+1)+” “+str(agj)) #can go upward

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi+1 and agvj==agj or agvi==agi and agvj==agj-1):

return agvi,agvj

else:

return -5,-5

elif(agi==1 and agj==0 or agi==2 and agj==0 or agi==3 and agj==0):

print(“\nyou can go at “+str(agi+1)+” “+str(agj)) #can go upward

print(“you can go at “+str(agi)+” “+str(agj+1)) #can move right

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi+1 and agvj==agj or agvi==agi and agvj==agj+1):
```

```
return agvi,agvj

else:

return -5,-5

elif(agi==0 and agj==3 or agi==1 and agj==3 or agi==2 and agj==3 or agi==3 and agj==3):

print(“you can go at “+str(agi+1)+” “+str(agj)) #can go upward

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

agvi=int(input(“Enter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi+1 and agvj==agj or agvi==agi and agvj==agj-1):

return agvi,agvj

else:

return -5,-5

elif(agi==3 and agj==1 or agi==3 and agj==2 or agi==3 and agj==3):

print(“\nyou can go at “+str(agi)+” “+str(agj+1)) #can go right

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

print(“you can go at “+str(agi-1)+” “+str(agj)) #can move downward

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))


if(agvi==agi and agvj==agj+1 or agvi==agi and agvj==agj-1 or agvi==agi-1 and agvj==agj):

return agvi,agvj

else:
```

```
return -5,-5

else:

print(“\nyou can go at “+str(agi)+” “+str(agj+1)) #can go right

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

print(“you can go at “+str(agi+1)+” “+str(agj)) #can move upward

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi and agvj==agj+1 or agvi==agi and agvj==agj-1 or agvi==agi+1 and agvj==agj):

return agvi,agvj

else:

return -5,-5


def checkinpreverse(agi,agj):

‘‘‘Function for checking input going in reverse direction to get back to original position’’’

if(agi==0 and agj==3):

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left


agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi and agvj==agj-1):

return agvi,agvj

else:

return -5,-5
```

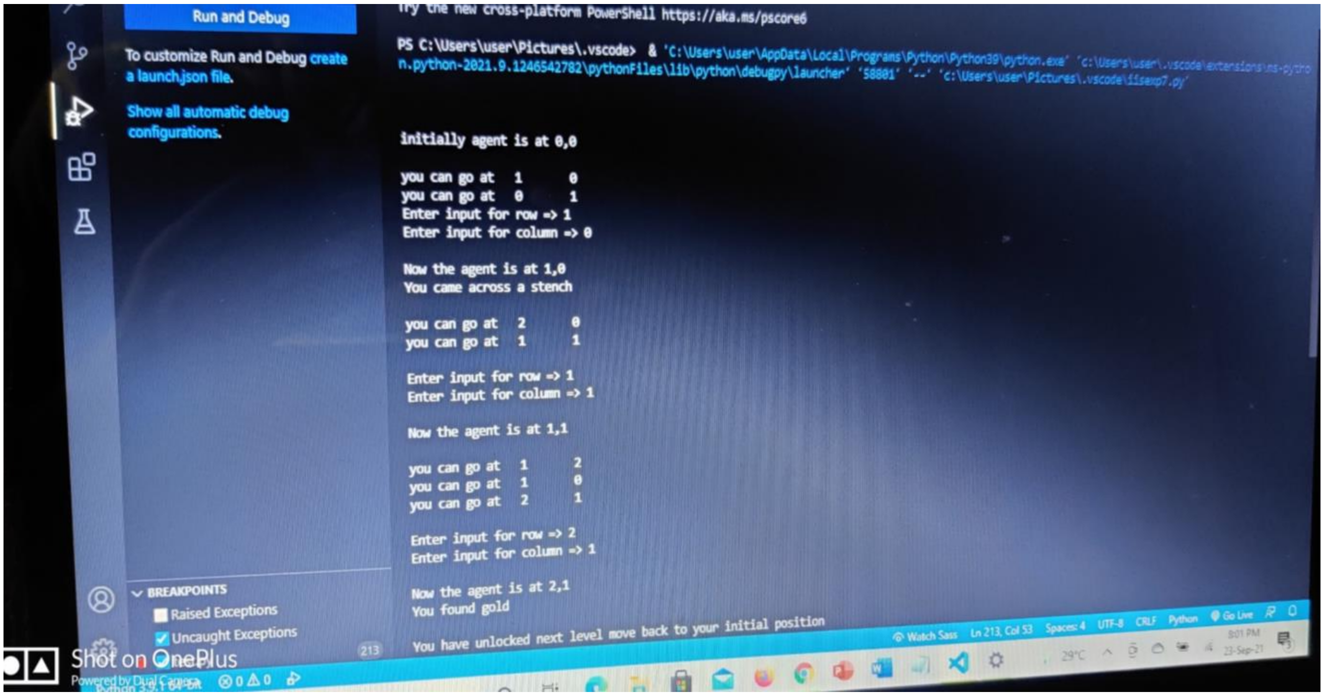
```
elif(agi==0 and agj==2 or agi==0 and agj==1):

print(“you can go at “+str(agi)+” “+str(agj+1)) #can go right

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi and agvj==agj-1 or agvi==agi and agvj==agj+1 ):

return agvi,agvj

else:

return -5,-5

elif(agi==1 and agj==0 or agi==2 and agj==0):

print(“\nyou can go at “+str(agi-1)+” “+str(agj)) #can go downward

print(“you can go at “+str(agi)+” “+str(agj+1)) #can move right


agvi=int(input(“\nEnter input for row =&gt; “))

agvj=int(input(“Enter input for column =&gt; “))

if(agvi==agi-1 and agvj==agj or agvi==agi and agvj==agj+1):

return agvi,agvj

else:

return -5,-5

elif(agi==1 and agj==3 or agi==2 and agj==3):

print(“you can go at “+str(agi-1)+” “+str(agj)) #can go downward

print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

agvi=int(input(“Enter input for row =&gt; “))
```

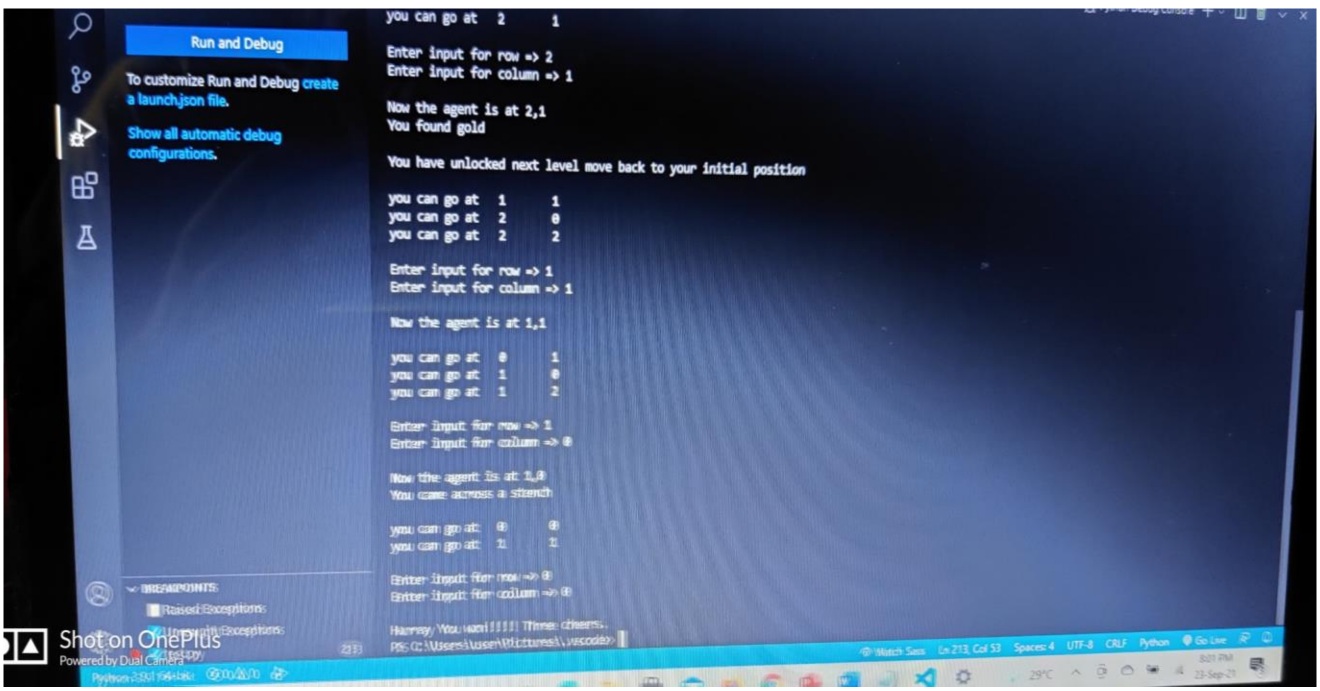
```
agvj=int(input(“Enter input for column =&gt; “))
if(agvi==agi-1 and agvj==agj or agvi==agi and agvj==agj-1):
return agvi,agvj
else:
return -5,-5
else:
print(“\nyou can go at “+str(agi-1)+” “+str(agj)) #can go downward
print(“you can go at “+str(agi)+” “+str(agj-1)) #can go left

print(“you can go at “+str(agi)+” “+str(agj+1)) #can go right
agvi=int(input(“\nEnter input for row =&gt; “))
agvj=int(input(“Enter input for column =&gt; “))
if(agvi==agi-1 and agvj==agj or agvi==agi and agvj==agj-1 or agvi==agi and agvj==agj+1):
return agvi,agvj
else:
return -5,-5

world=[ [0,5,7,5],
[9,0,8,0],
[-1,6,7,8],
[9,0,8,7] ] #declaration of a world

agi,agj=0,0 #initial agent position
```

```
print(“\n\n\ninitially agent is at “+str(agi)+”,”+str(agj))

print(“\nyou can go at “+str(agi+1)+” “+str(agj))

print(“you can go at “+str(agi)+” “+str(agj+1))


agvi=int(input(“Enter input for row =&gt; “))


agvj=int(input(“Enter input for column =&gt; “)) #taking row and column values

if(agvi==1 and agvj==0 or agvi==0 and agvj==1):

agi,agj=learnagent(world,agvi,agvj) #if input valid calling learn agent function

else:

print(“Not valid”)


while(agi&gt;=0):

agvi,agvj=checkinp(agi,agj)

if(agvi!=-5 and agvj!=-5):

agi,agj=learnagent(world,agvi,agvj)

else:

print(“\nNot valid”)


if(agi==-5):

print(“\nGame over Sorry try next time!!!”)

else:

print(“\nYou have unlocked next level move back to your initial position”) #acquired gold
```

```
agi,agj=2,1 #implementation of reverse logic

while(agi&gt;=0):
agvi,agvj=checkinpreverse(agi,agj)
if(agvi==0 and agvj==0):
agi,agj=-4,-4
elif(agvi!=-5 and agvj!=-5):
agi,agj=learnagent(world,agvi,agvj)
else:
print(“\nNot valid”)

if(agi==-5):
print(“\nYou were really close but unfortunately you failed!!! Try next time”)
else:
print(“\nHurray You won!!!!! Three cheers.”)
```

Output:





Result and Discussion: 1) We have implemented Wumpus world problem using python.

2) Wumpus world makes use of knowledge-based agent to play.

3) It is a simple demonstration of how a knowledge-based agent works in real life.

Learning Outcomes: The student should have the ability to

LO1: identify a problem which can be solved using informed search methods.

LO2: implement informed search methods.

LO3: describe properties of informed search algorithm.

LO4: identify advantage and disadvantage of the algorithm.

Course Outcomes: Upon completion of the course, students will be able to build knowledge based agent and make inferences to answer the queries posed.

Conclusion: We have understood about Wumpus agent game. We have implemented the code on it and we are able to apply in real life.

Viva Questions:

1. State PEAS descriptor for Wumpus World problem.
2. State inference Rules.
3. State the steps of conversion to CNF.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				