Date: **Ex No:** **12.01.2022**	**Title of the Lab** **Agent and Real World Problems**	**Name: MAINAK CHAUDHURI** **Registration Number: RA1911027010039** **Section: N1** **Lab Batch: 2** **Day Order: 3**

AIM: Wumpus World Problem,

Description of the Concept or Problem given:

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Manual Solution

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus. At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

Agent's second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room? Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].

Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Program Implementation [Coding]

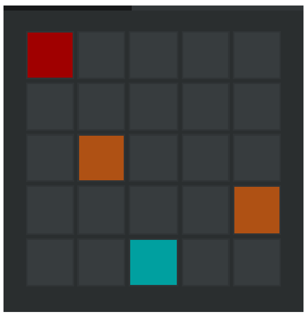
```
#Download and extract all necessary files

!rm -rf /content/*

!wget https://github.com/aimacode/aima-python/archive/master.zip 2>/dev/null

!unzip -q master.zip

!mv aima-python-master/* /content

!wget https://github.com/aimacode/aima-
data/archive/f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86.zip 2>/dev/null

!unzip -q f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86.zip

!rm -rf aima-data

!mv aima-data-f6cbea61ad0c21c6b7be826d17af5a8d3a7c2c86 aima-data

#Install Libraries

!pip install ipythonblocks 2>/dev/null


from agents import *


class BlindDog(Agent):

    def eat(self, thing):
        print("Dog: Ate food at {}.".format(self.location))

    def drink(self, thing):
        print("Dog: Drank water at {}.".format( self.location))

dog = BlindDog()
class Food(Thing):
    pass

class Water(Thing):
    pass

class Park(Environment):
    def percept(self, agent):
        '''return a list of things that are in our agent's location'''
        things = self.list_things_at(agent.location)
        return things

    def execute_action(self, agent, action):
        '''changes the state of the environment based on what the agent does.'''
        if action == "move down":
            print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, agent.location))
            agent.movedown()
        elif action == "eat":
            items = self.list_things_at(agent.location, tclass=Food)
            if len(items) != 0:
                if agent.eat(items[0]): #Have the dog eat the first item
                    print('{} ate {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0]) #Delete it from the Park after.
        elif action == "drink":
            items = self.list_things_at(agent.location, tclass=Water)
            if len(items) != 0:
                if agent.drink(items[0]): #Have the dog drink the first item
                    print('{} drank {} at location: {}'
                          .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                    self.delete_thing(items[0]) #Delete it from the Park after.

    def is_done(self):
        '''By default, we're done when we can't find a live agent,
        but to prevent killing our cute dog, we will stop before itself - when there is no more
food or water'''
        no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in
self.things)
        dead_agents = not any(agent.is_alive() for agent in self.agents)
        return dead_agents or no_edibles
class BlindDog(Agent):
    location = 1

    def movedown(self):
        self.location += 1

    def eat(self, thing):
        '''returns True upon success or False otherwise'''
        if isinstance(thing, Food):
            return True
        return False

    def drink(self, thing):
        ''' returns True upon success or False otherwise'''
        if isinstance(thing, Water):
            return True
        return False

def program(percepts):
    '''Returns an action based on it's percepts'''
    for p in percepts:
        if isinstance(p, Food):
            return 'eat'
        elif isinstance(p, Water):
            return 'drink'
    return 'move down'
    park = Park()
    dog = BlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, 1)
    park.add_thing(dogfood, 5)
    park.add_thing(water, 7)

    park.run(5)
    park.add_thing(water, 15)
    park.run(10)
    class Park2D(XYEnvironment):
        def percept(self, agent):
            '''return a list of things that are in our agent's location'''
            things = self.list_things_at(agent.location)
            return things

        def execute_action(self, agent, action):
            '''changes the state of the environment based on what the agent does.'''
            if action == "move down":
                print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action,
agent.location))
                agent.movedown()
            elif action == "eat":
                items = self.list_things_at(agent.location, tclass=Food)
                if len(items) != 0:
                    if agent.eat(items[0]): #Have the dog eat the first item
                        print('{} ate {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0]) #Delete it from the Park after.
            elif action == "drink":
                items = self.list_things_at(agent.location, tclass=Water)
                if len(items) != 0:
                    if agent.drink(items[0]): #Have the dog drink the first item
                        print('{} drank {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0]) #Delete it from the Park after.


        def is_done(self):
            '''By default, we're done when we can't find a live agent,
            but to prevent killing our cute dog, we will stop before itself - when there is no more
food or water'''
            no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in
self.things)
            dead_agents = not any(agent.is_alive() for agent in self.agents)
            return dead_agents or no_edibles


    class BlindDog(Agent):
        location = [0,1] # change location to a 2d value
        direction = Direction("down") # variable to store the direction our dog is facing
        def movedown(self):
            self.location[1] += 1

        def eat(self, thing):
            '''returns True upon success or False otherwise'''
            if isinstance(thing, Food):
                return True
            return False

        def drink(self, thing):
            ''' returns True upon success or False otherwise'''
            if isinstance(thing, Water):
                return True
            return False

    def program(percepts):
        '''Returns an action based on it's percepts'''
        for p in percepts:
            if isinstance(p, Food):
                return 'eat'
            elif isinstance(p, Water):
                return 'drink'
        return 'move down'
    park = Park2D(5,20) # park width is set to 5, and height to 20
    dog = BlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, [0,1])
    park.add_thing(dogfood, [0,5])
    park.add_thing(water, [0,7])
    morewater = Water()
    park.add_thing(morewater, [0,15])
    park.run(20)
    from random import choice

    turn = False # global variable to remember to turn if our dog hits the boundary
    class EnergeticBlindDog(Agent):
        location = [0,1]
        direction = Direction("down")

        def moveforward(self, success=True):
            '''moveforward possible only if success (ie valid destination location)'''
            global turn
            if not success:
                turn = True # if edge has been reached, remember to turn
                return
            if self.direction.direction == Direction.R:
                self.location[0] += 1
            elif self.direction.direction == Direction.L:
                self.location[0] -= 1
            elif self.direction.direction == Direction.D:
                self.location[1] += 1
            elif self.direction.direction == Direction.U:
                self.location[1] -= 1

        def turn(self, d):
            self.direction = self.direction + d

        def eat(self, thing):
            '''returns True upon success or False otherwise'''
            if isinstance(thing, Food):
                return True
            return False

        def drink(self, thing):
            ''' returns True upon success or False otherwise'''
            if isinstance(thing, Water):
                return True
            return False

    def program(percepts):
        '''Returns an action based on it's percepts'''
        global turn
        for p in percepts: # first eat or drink - you're a dog!
            if isinstance(p, Food):
                return 'eat'
            elif isinstance(p, Water):
                return 'drink'
        if turn: # then recall if you were at an edge and had to turn
            turn = False
            choice = random.choice((1,2));
        else:
            choice = random.choice((1,2,3,4)) # 1-right, 2-left, others-forward
        if choice == 1:
            return 'turnright'
        elif choice == 2:
            return 'turnleft'
        else:
            return 'moveforward'
```

```
class Park2D(XYEnvironment):
def percept(self, agent):
    '''return a list of things that are in our agent's location'''
    things = self.list_things_at(agent.location)
    return things

def execute_action(self, agent, action):
    '''changes the state of the environment based on what the agent does.'''
    if action == 'turnright':
        print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action,
agent.location))
        agent.turn(Direction.R)
    elif action == 'turnleft':
        print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action,
agent.location))
        agent.turn(Direction.L)
    elif action == 'moveforward':
        loc = copy.deepcopy(agent.location) # find out the target location
        if agent.direction.direction == Direction.R:
            loc[0] += 1
        elif agent.direction.direction == Direction.L:
            loc[0] -= 1
        elif agent.direction.direction == Direction.D:
            loc[1] += 1
        elif agent.direction.direction == Direction.U:
            loc[1] -= 1
        if self.is_inbounds(loc):# move only if the target is a valid location
            print('{} decided to move {}wards at location: {}'.format(str(agent)[1:-1],
agent.direction.direction, agent.location))
            agent.moveforward()
        else:
```

```
                    print('{} decided to move {}wards at location: {}, but
couldn\'t'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
                    agent.moveforward(False)
            elif action == "eat":
                items = self.list_things_at(agent.location, tclass=Food)
                if len(items) != 0:
                    if agent.eat(items[0]):
                        print('{} ate {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])
            elif action == "drink":
                items = self.list_things_at(agent.location, tclass=Water)
                if len(items) != 0:
                    if agent.drink(items[0]):
                        print('{} drank {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])

        def is_done(self):
            '''By default, we're done when we can't find a live agent,
            but to prevent killing our cute dog, we will stop before itself - when there is no more
food or water'''
            no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in
self.things)
            dead_agents = not any(agent.is_alive() for agent in self.agents)
            return dead_agents or no_edibles
    park = Park2D(3,3)
    dog = EnergeticBlindDog(program)
    dogfood = Food()
    water = Water()
    park.add_thing(dog, [0,0])
    park.add_thing(dogfood, [1,2])
    park.add_thing(water, [2,1])
    morewater = Water()
    park.add_thing(morewater, [0,2])
    print("dog started at [0,0], facing down. Let's see if he found any food or water!")
    park.run(20)
    class GraphicPark(GraphicEnvironment):
        def percept(self, agent):
            '''return a list of things that are in our agent's location'''
            things = self.list_things_at(agent.location)
            return things

        def execute_action(self, agent, action):
            '''changes the state of the environment based on what the agent does.'''
            if action == 'turnright':
                print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, 
agent.location))
                agent.turn(Direction.R)
            elif action == 'turnleft':
                print('{} decided to {} at location: {}'.format(str(agent)[1:-1], action, 
agent.location))
                agent.turn(Direction.L)
            elif action == 'moveforward':
                loc = copy.deepcopy(agent.location) # find out the target location
                if agent.direction.direction == Direction.R:
                    loc[0] += 1
                elif agent.direction.direction == Direction.L:
                    loc[0] -= 1
                elif agent.direction.direction == Direction.D:
                    loc[1] += 1
                elif agent.direction.direction == Direction.U:
                    loc[1] -= 1
                if self.is_inbounds(loc):# move only if the target is a valid location
                    print('{} decided to move {}wards at location: {}'.format(str(agent)[1:-1],
agent.direction.direction, agent.location))
                    agent.moveforward()
                else:
                    print('{} decided to move {}wards at location: {}, but
couldn\'t'.format(str(agent)[1:-1], agent.direction.direction, agent.location))
                    agent.moveforward(False)
            elif action == "eat":
                items = self.list_things_at(agent.location, tclass=Food)
                if len(items) != 0:
                    if agent.eat(items[0]):
                        print('{} ate {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])
            elif action == "drink":
                items = self.list_things_at(agent.location, tclass=Water)
                if len(items) != 0:
                    if agent.drink(items[0]):
                        print('{} drank {} at location: {}'
                              .format(str(agent)[1:-1], str(items[0])[1:-1], agent.location))
                        self.delete_thing(items[0])


        def is_done(self):
            '''By default, we're done when we can't find a live agent,
            but to prevent killing our cute dog, we will stop before itself - when there is no more
food or water'''
            no_edibles = not any(isinstance(thing, Food) or isinstance(thing, Water) for thing in
self.things)
            dead_agents = not any(agent.is_alive() for agent in self.agents)
            return dead_agents or no_edibles
```

```
from ipythonblocks import BlockGrid


park = GraphicPark(5,5, color={'EnergeticBlindDog': (200,0,0), 'Water': (0, 200, 200),
'Food': (230, 115, 40)})
dog = EnergeticBlindDog(program)
dogfood = Food()
water = Water()
park.add_thing(dog, [0,0])
park.add_thing(dogfood, [1,2])
park.add_thing(water, [0,1])
morewater = Water()
morefood = Food()
park.add_thing(morewater, [2,4])
park.add_thing(morefood, [4,3])
print("dog started at [0,0], facing down. Let's see if he found any food or water!")
park.run(20)


from ipythonblocks import BlockGrid
from agents import *


color = {"Breeze": (225, 225, 225),
        "Pit": (0,0,0),
        "Gold": (253, 208, 23),
        "Glitter": (253, 208, 23),
        "Wumpus": (43, 27, 23),
        "Stench": (128, 128, 128),
        "Explorer": (0, 0, 255),
        "Wall": (44, 53, 57)
        }


def program(percepts):
    '''Returns an action based on it's percepts'''
    print(percepts)
    return input()
w = WumpusEnvironment(program, 7, 7)
grid = BlockGrid(w.width, w.height, fill=(123, 234, 123))

def draw_grid(world):
    global grid
    grid[:] = (123, 234, 123)
    for x in range(0, len(world)):
        for y in range(0, len(world[x])):
            if len(world[x][y]):
                grid[y, x] = color[world[x][y][-1].__class__.__name__]

def step():
    global grid, w
    draw_grid(w.get_world())
    grid.show()
    w.step()
```

Screenshots of the Outputs:



Mainak Chaudhuri

Signature of the Student

MAINAK CHAUHDURI