Project 1 Intelligent Agents - Reflex Agents

In this project, we were tasked with creating 3 types of intelligent reflex agents. Below is the code setup I used to implement each agent.

```
from agents import *
from notebook import psource
```

```
class Food(Thing):
    pass
```

```
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)
        loc = copy.deepcopy(agent.location) # find out the
target location
        #Check if agent is about to bump into a wall
        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 not self.is_inbounds(loc):
            things.append(Bump())
```

```
        return things

    def execute_action(self, agent, action):
        '''changes the state of the environment based on wh
at 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':
            print('{} decided to move {}wards at location:
{}'.format(str(agent)[1:-1], agent.direction.direction, age
nt.location))
            agent.moveforward()
        elif action == "eat":
            items = self.list_things_at(agent.location, tcl
ass=Food)
            if len(items) != 0:
                if agent.eat(items[0]):
                    print('{} ate {} at location: {}'
                          .format(str(agent)[1:-1], str(ite
ms[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 b
efore itself - when there is no more food'''
        no_edibles = not any(isinstance(thing, Food) for th
ing in self.things)
        dead_agents = not any(agent.is_alive() for agent in
self.agents)
        return dead_agents or no_edibles
```

```
#Variables for All
food = 10
moves = 100
```

```
from random import choice

class EnergeticBlindDog(Agent):
    location = [0,1]
    direction = Direction("down")
    performacnce = 0
    def moveforward(self, success=True):
        '''moveforward possible only if success (i.e. valid
destination location)'''
        if not success:
            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):
            self.performacnce += 1
            return True
        return False
```

Below Is the implementation of each Reflex Agent

Simple Reflex Agent

This Agent has no memory or the ability to take random actions, the only thing its able to do is perform a set action when encountering some percept. In this case a bump, where it turns left. As a result in a square grid it will just coast around the outside.

```
#Simple Reflex Agent Program
def program(percepts):
    '''Returns an action based on it's percepts'''

    for p in percepts: # first eat or drink - you're a dog!
        if isinstance(p, Food):
            return 'eat'
        if isinstance(p,Bump): # then check if you are at a
n edge and have to turn
            turn = False
            #print("bump!")
            choice = 2
        else:
            choice = 3 # 1-right, 2-left, others-forward
    if choice == 1:
        return 'turnright'
    elif choice == 2:
        return 'turnleft'
    else:
        return 'moveforward'
```

Simple Random Reflex Agent

This agent takes random actions, the random.randint(0,10) defines it's behavior. If this random number is a 1 or a 2 it will turn else it will move forward. This is a 20% chance to turn and an 80% chance to move forward.
This agents behavior is random and as such we cannot determine how it will specifically move before hand.

```
#Simple Random Agent Program
def programRandom(percepts):
    '''Returns an action based on it's percepts'''

    for p in percepts: # first eat or drink - you're a dog!
        if isinstance(p, Food):
            return 'eat'
        if isinstance(p,Bump): # then check if you are at a
n edge and have to turn
            turn = False
            #print("bump!")
            choice = 2
        else:
            choice = random.randint(0,10) # 1-right, 2-lef
t, others-forward
    if choice == 1:
        return 'turnright'
    elif choice == 2:
        return 'turnleft'
    else:
        return 'moveforward'
```

Limited Memory Reflex Agent

This agent has some memory of his previous actions and percepts, such as how many bumps he has recently run into and previous action at a bump through current state. With this the reflex agent is able to take more complex actions and fill any grid like spaces. When it hits a bump but doesn't detect

another bump while going through some set actions it will continue to fill the area, when it hits a bump in either a corner or a pocket in non square grids it will shift its direction and continue trying different states to get out and continue filling the area.

```
#Limited Memory Agent Program
import queue
dogMemory = {True:[1,3,1],False:[2,3,2],'currentState':Fals
e,'StatePos':0,'bump':False,'bumpcount':0}
#Simple Random Agent Program
def programLimited(percepts):
    '''Returns an action based on it's percepts'''
    shift = 0
    for p in percepts: # first eat or drink - you're a dog!
        if isinstance(p, Food):
            return 'eat'
        if isinstance(p,Bump): # then check if you are at a
n edge and have to turn
            turn = False
            dogMemory['bump'] = True
            dogMemory['bumpcount'] +=1
            print("bump!",dogMemory['bumpcount'],dogMemory
['currentState'])
        else:
            choice = 3
    if dogMemory['bump']:
        if dogMemory['StatePos'] > 2:
            dogMemory['StatePos'] = 0
            dogMemory['bump'] = False
            dogMemory['bumpcount'] = 0
            dogMemory['currentState'] = not dogMemory['curr
entState']
        elif dogMemory['bumpcount'] >= 2:
            shift = 1
            dogMemory['bumpcount'] = 0
        else:
            choice = dogMemory[dogMemory['currentState']][d
```

```
ogMemory['StatePos']]
            dogMemory['StatePos'] +=1
        if shift > 0:
            choice = 2
            shift +=1
        if shift > 2:
            shift = 0
    if choice == 1:
        return 'turnright'
    elif choice == 2:
        return 'turnleft'
    else:
        return 'moveforward'
```

Measuring Agents Performance

To measure performance I collected information about the percentage of all food they collected compared to the food in the room. Without comparing the food collected to the food in the room we can run into the issue of the agent potentially abusing the metric to score better, in ways such as going away from the environment to find food to eat or finding a way to create more food instead of moving. This is bad as our goal is to have the dog eat as much possible food from the environment and as such the performance criteria must reflect that.

Performance calculation: Food Eaten - Food Left

This data was collected from 100 runs with a random starting position and 15 food on the grid with 100 moves allocated to each agent.

Agent	Performance	Approximate Amount
Simple Reflex Agent	-491	5/15
Simple Random Reflex Agent	-273	6/15
Limited Memory Reflex Agent	259	9/15

As we can see the both the simple reflex agent and the simple random reflex agent both performed quite poorly compared to the limited memory reflex agent. This was the expected outcome as the limited memory agent is much more consistent in visiting every space therefore eating the most food compared to the other agents.

Final Notes and Possible Improvements

Overall I achieved an expected result from my agents and was able to increase my understanding of these reflex agents. There was some issues with my implementation. It is possible some of my implementations would not work on non-square areas and could get stuck in a corner, another possibility for improvement is changing the implementation of the Limited Memory Agent. My implementation will not be as efficient as a space filling curve as it will often visit previously visited spaces.