Implementation of Dec-POMDP Environment in Spann 2006

Baiting Zhu

Table of Contents

1. DEC-POMDP vs. POMDP
2. Assumptions of the experiment environment
3. Transition Function
4. Reward Function
5. Observation Function
6. Further Improvements

DEC-POMDP vs. POMDP

What makes a POMDP “decentralized”?

- More than 1 agent

  agentList = [“agentOne”, “agentTwo”, …]

- Agents taking actions simultaneously and are independent of each other

  allAgentsActionTable = {“agentOne”: (1, 0), “agentTwo”: (0, 0), …}

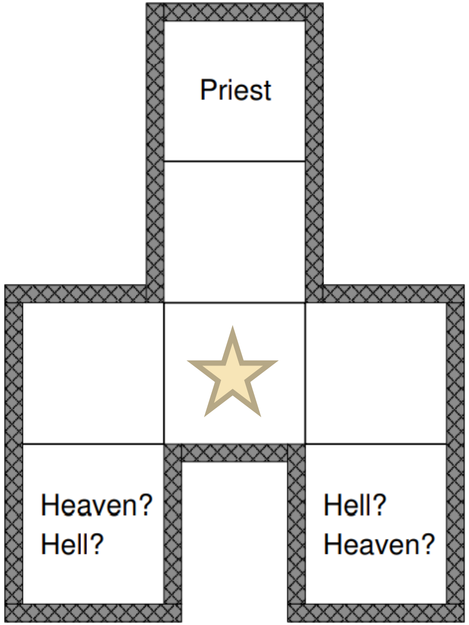
- Agents have (limited) observations

  allAgentsObsTable = {“agentOne”: “left”, “agentTwo”: “heaven-right”, …}

- Agents may communicate (share observations) at a cost
- Each agent only considers its local state + shared information about the environment

  NOT local states of other agents

Spaan’s Environment



Two agents start from the star and move independently to the left, right, up, down, or stay. Any movement has -0.1 cost.

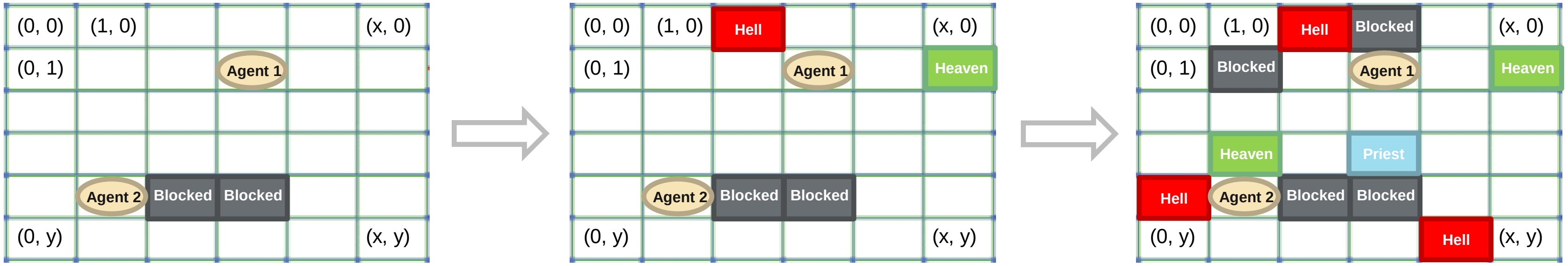
One of the bottom-left or bottom-right grid is Heaven, and the other one is Hell. This is unknown to the agents at the start.

Agents know which one is the Heaven when visiting the Priest. Doing so results in a -2 cost.

Once agents meet up in a grid that is Heaven or Hell, the game ends. With a +10 reward or -10 punishment.

Our Environment

- Any number of $\boldsymbol{n}$ **agents** may take actions simultaneously in a **2D rectangular** environment of size $\boldsymbol{x} \times \boldsymbol{y}$
- Any grid within the environment may be “**Blocked”** (unable to enter nor start from)
- Any number of **“Heaven”** and **“Hell”** may exist in unblocked grids, but they cannot be in the same grid. Each “Heaven” / “Hell” may also have unique **reward** / **punishment** values
- Any number of “**Priest”** may exist in unblocked grids that are not Heaven nor Hell. Each “Priest” may have unique cost



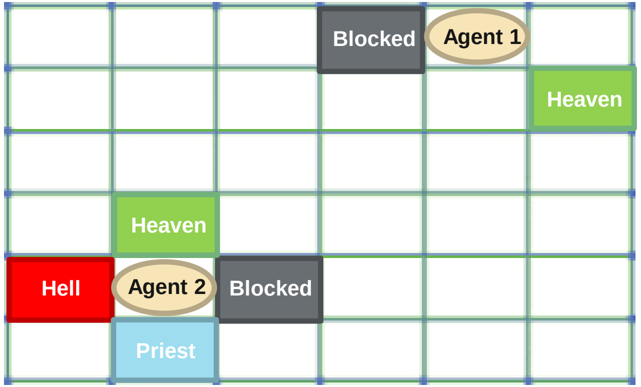
(0, 0)	(1, 0)				(x, 0)
(0, 1)			Agent 1		
	Agent 2	Blocked	Blocked		
(0, y)					(x, y)

(0, 0)	(1, 0)	Hell			(x, 0)
(0, 1)			Agent 1		Heaven
	Agent 2	Blocked	Blocked		
(0, y)					(x, y)

(0, 0)	(1, 0)	Hell	Blocked		(x, 0)
(0, 1)	Blocked		Agent 1		Heaven
	Heaven		Priest		
Hell	Agent 2	Blocked	Blocked		
(0, y)				Hell	(x, y)

Game Movements

- Each agent must be in an unblocked grid in the map at any time stamp t
- Each agent may take actions $(\mathbf{1},\mathbf{0})$ $(\mathbf{0},\mathbf{1})$ $(-\mathbf{1},\mathbf{0})$ $(\mathbf{0},-\mathbf{1})$ $(\mathbf{0},\mathbf{0})$ at each time stamp t, correspondingly representing **Move Right**, **Move Down**, **Move Left**, **Move Up**, and **Stay**
- When an action is leading the agent **out of boundary** or **into a Blocked grid**, agent's action result in no movement



Block & Boundary are not reachable:

Agent 1, move left: location unchanged, due to block
Agent 1, move up: location unchanged, due to boundary

Heaven / Hell / Priest can be reached:

Agent 2, move up: location changes
Agent 2, move up: location changes
Agent 2, move up: location changes
Agent 2, move up: location unchanged

Game Reward and Punishment

allAgentsNormalCost = {“agentOne”: -0.1, “agentTwo”: -0.2, …}

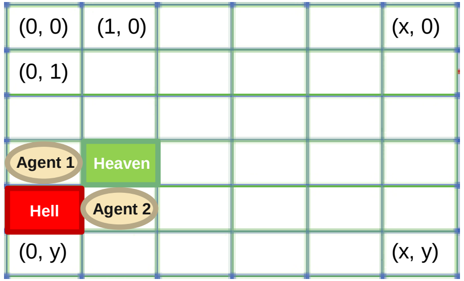
- No matter the action an agent chooses to do (including stay), and no matter the consequence of the action, agent pays a **normal cost**. This cost may be different for each agent
- When all agents meet in **the same Heaven or Hell**, game ends. All agents take the award / punishment
- When game ends, further actions result in no cost nor movement
- Being in different Heaven or Hell doesn’t lead to an end of the game
- *Other than normal cost and ending award / punishment, each agent may experience different reward / punishment along the way

<u>Agent 1 move right; Agent 2 moves up</u>

-> Happy Ending

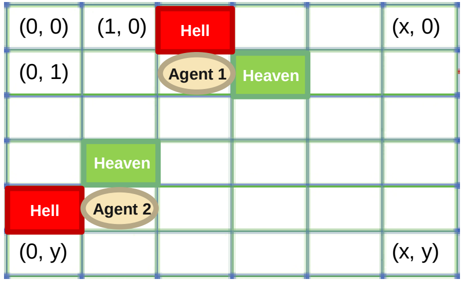
<u>Agent 1 move down; Agent move left</u>

-> Bad Ending



(0, 0)	(1, 0)				(x, 0)
(0, 1)					
Agent 1	Heaven				
Hell	Agent 2				
(0, y)					(x, y)

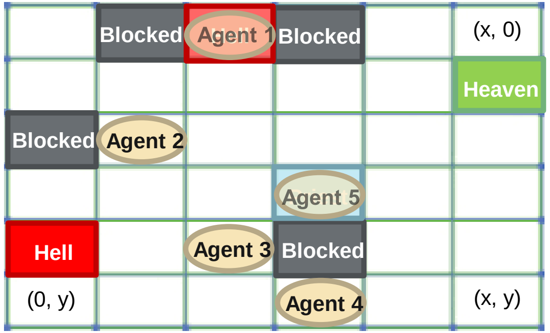
<u>Game cannot end in one step</u>, because agent cannot meet up in the SAME Heaven or Hell



(0, 0)	(1, 0)	Hell			(x, 0)
(0, 1)		Agent 1	Heaven		
	Heaven				
Hell	Agent 2				
(0, y)					(x, y)

Game Observations and Priest

- If an agent is not in a Priest grid, an agent observes whether there is a wall on its left and/or right. Note that this only relates to the horizontal (x) direction of the map, but not agent’s orientation. May observe **“left”, “right”, “none”, or “both”**
- If an agent is in a Priest grid, it takes a cost and knows the location of each Heaven. Different priest may have different costs



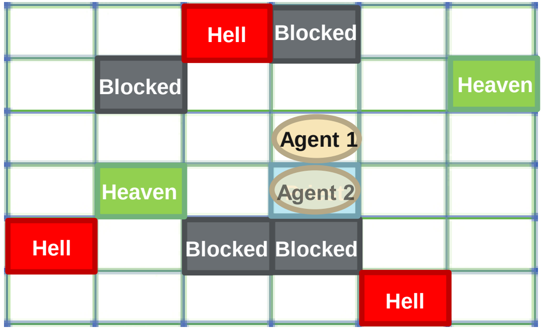
Agent 1: observes “both”
Agent 2: observes “left”
Agent 3: observes “right”
Agent 4: observes “neither“

$-x$
wall-obs direction
$+x$

Agent 5: observes location of each Heaven

More Rules about Priest Cost

- If agent **stays in the Priest grid**, or its actions has **no resulting change in location** (into boundary or block), it **doesn't** take the Priest cost again
- Getting out from the Priest grid then **re-enters will result** in the same Priest cost
- Otherwise, the Priest grid should be all the same as a normal grid



		Hell	Blocked		
	Blocked				Heaven
			Agent 1		
	Heaven		Agent 2		
Hell		Blocked	Blocked		
				Hell	

Agent 1, move down: Priest cost paid (visiting)

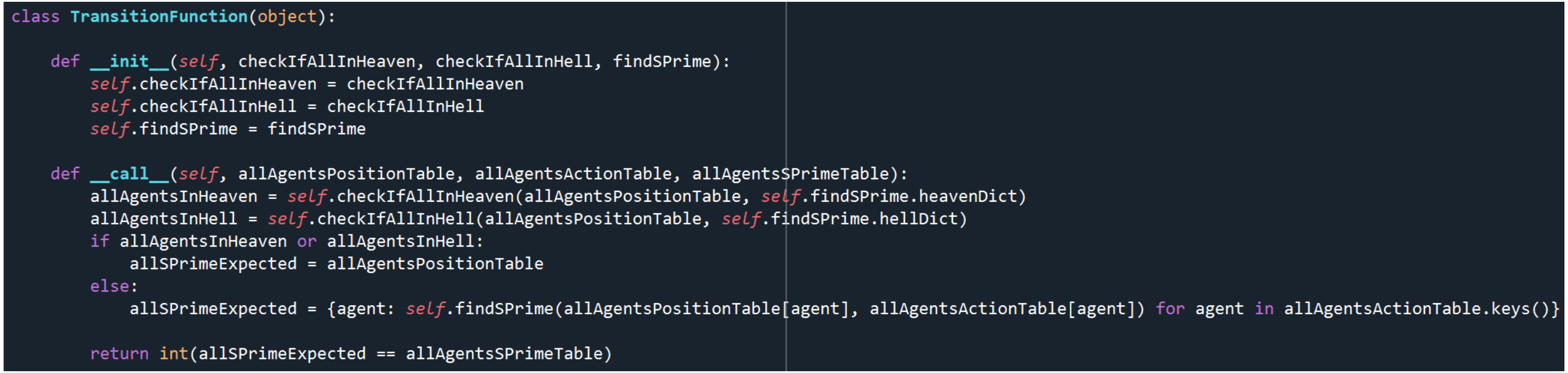
Agent 2, move down: NO cost (blocked ≡ staying)

Agent 2, stays: NO cost (staying)

Agent 2, move left then right: Priest cost paid (re-entering ≡ visiting)

Transition Function (All Agents)

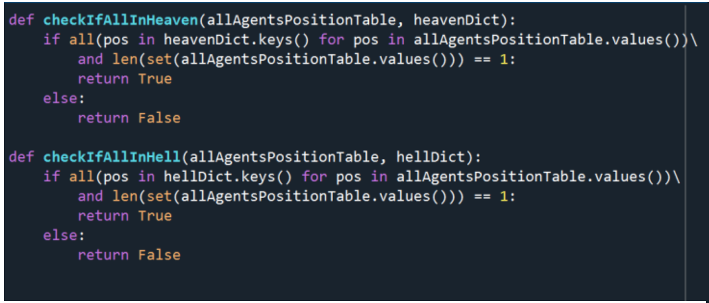
- $p(s'|s, a)$ where s', s, a are vectors representing the resulting states, original states, and actions of all agents



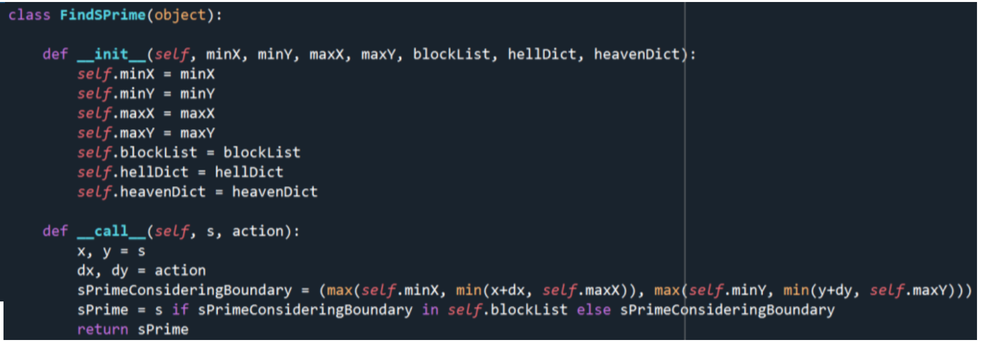
```python
class TransitionFunction(object):

    def __init__(self, checkIfAllInHeaven, checkIfAllInHell, findSPrime):
        self.checkIfAllInHeaven = checkIfAllInHeaven
        self.checkIfAllInHell = checkIfAllInHell
        self.findSPrime = findSPrime

    def __call__(self, allAgentsPositionTable, allAgentsActionTable, allAgentsSPrimeTable):
        allAgentsInHeaven = self.checkIfAllInHeaven(allAgentsPositionTable, self.findSPrime.heavenDict)
        allAgentsInHell = self.checkIfAllInHell(allAgentsPositionTable, self.findSPrime.hellDict)
        if allAgentsInHeaven or allAgentsInHell:
            allSPrimeExpected = allAgentsPositionTable
        else:
            allSPrimeExpected = {agent: self.findSPrime(allAgentsPositionTable[agent], allAgentsActionTable[agent]) for agent in allAgentsActionTable.keys()}

        return int(allSPrimeExpected == allAgentsSPrimeTable)
```

Transition Function (One Agent)

- $p(s'|s, a)$ where s', s, a are vectors representing the resulting states, original states, and actions of all agents



```python
def checkIfAllInHeaven(allAgentsPositionTable, heavenDict):
    if all(pos in heavenDict.keys() for pos in allAgentsPositionTable.values())\
        and len(set(allAgentsPositionTable.values())) == 1:
        return True
    else:
        return False

def checkIfAllInHell(allAgentsPositionTable, hellDict):
    if all(pos in hellDict.keys() for pos in allAgentsPositionTable.values())\
        and len(set(allAgentsPositionTable.values())) == 1:
        return True
    else:
        return False
```



```python
class FindSPrime(object):

    def __init__(self, minX, minY, maxX, maxY, blockList, hellDict, heavenDict):
        self.minX = minX
        self.minY = minY
        self.maxX = maxX
        self.maxY = maxY
        self.blockList = blockList
        self.hellDict = hellDict
        self.heavenDict = heavenDict

    def __call__(self, s, action):
        x, y = s
        dx, dy = action
        sPrimeConsideringBoundary = (max(self.minX, min(x+dx, self.maxX)), max(self.minY, min(y+dy, self.maxY)))
        sPrime = s if sPrimeConsideringBoundary in self.blockList else sPrimeConsideringBoundary
        return sPrime
```

heavenDict = {(1, 2): 10, (2, 3): 9}

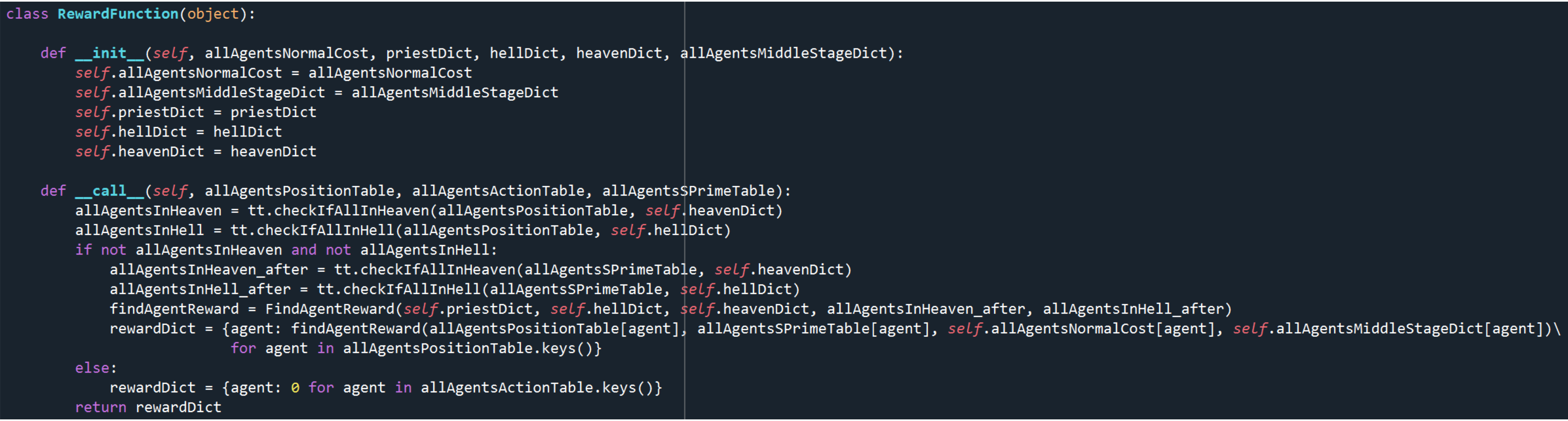
allAgents[…]Table = {“agentOne”: …, “agentTwo”: …}

hellDict = {(1, 1): -5, (4, 3): -9.6}

hellDict = [(1, 0), (4, 6), (7, 2)]

Reward Function (All Agents)

- $R(s, a, s')$ where s, a, s' are vectors representing the, original states, actions, and resulting states of all agents

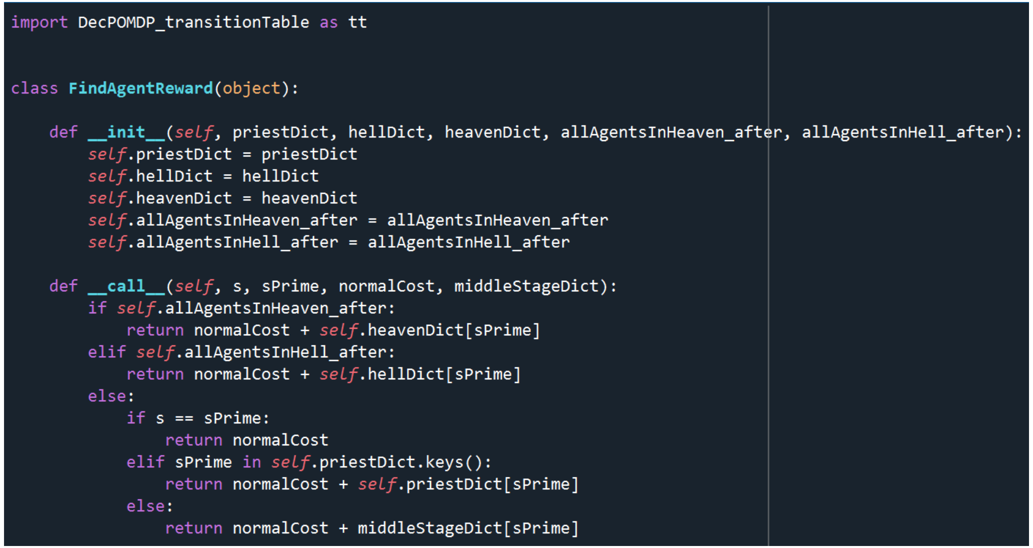


```
class RewardFunction(object):

    def __init__(self, allAgentsNormalCost, priestDict, hellDict, heavenDict, allAgentsMiddleStageDict):
        self.allAgentsNormalCost = allAgentsNormalCost
        self.allAgentsMiddleStageDict = allAgentsMiddleStageDict
        self.priestDict = priestDict
        self.hellDict = hellDict
        self.heavenDict = heavenDict

    def __call__(self, allAgentsPositionTable, allAgentsActionTable, allAgentsSPrimeTable):
        allAgentsInHeaven = tt.checkIfAllInHeaven(allAgentsPositionTable, self.heavenDict)
        allAgentsInHell = tt.checkIfAllInHell(allAgentsPositionTable, self.hellDict)
        if not allAgentsInHeaven and not allAgentsInHell:
            allAgentsInHeaven_after = tt.checkIfAllInHeaven(allAgentsSPrimeTable, self.heavenDict)
            allAgentsInHell_after = tt.checkIfAllInHell(allAgentsSPrimeTable, self.hellDict)
            findAgentReward = FindAgentReward(self.priestDict, self.hellDict, self.heavenDict, allAgentsInHeaven_after, allAgentsInHell_after)
            rewardDict = {agent: findAgentReward(allAgentsPositionTable[agent], allAgentsSPrimeTable[agent], self.allAgentsNormalCost[agent], self.allAgentsMiddleStageDict[agent])\
                          for agent in allAgentsPositionTable.keys()}
        else:
            rewardDict = {agent: 0 for agent in allAgentsActionTable.keys()}
        return rewardDict
```

Total reward = sum of reward of all agents → e.g.: sum(list(rewardDict.values()))

Reward Function (One Agent)

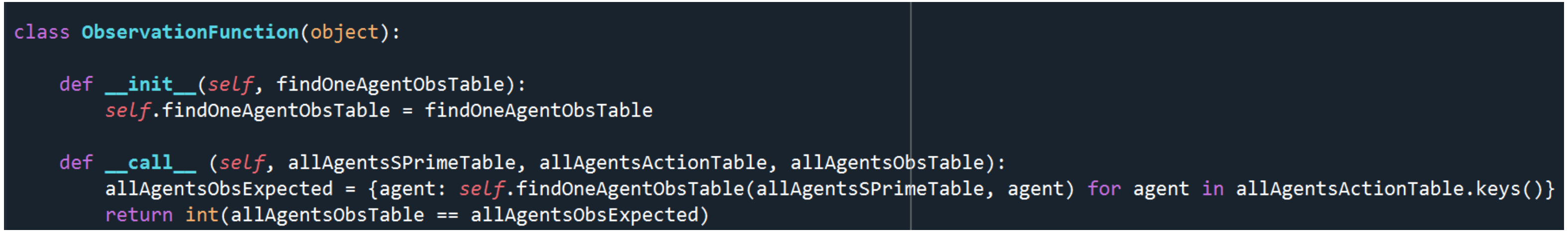
- $R(s, a, s')$ where s, a, s' are vectors representing the, original states, actions, and resulting states of all agents



```python
import DecPOMDP_transitionTable as tt


class FindAgentReward(object):

    def __init__(self, priestDict, hellDict, heavenDict, allAgentsInHeaven_after, allAgentsInHell_after):
        self.priestDict = priestDict
        self.hellDict = hellDict
        self.heavenDict = heavenDict
        self.allAgentsInHeaven_after = allAgentsInHeaven_after
        self.allAgentsInHell_after = allAgentsInHell_after

    def __call__(self, s, sPrime, normalCost, middleStageDict):
        if self.allAgentsInHeaven_after:
            return normalCost + self.heavenDict[sPrime]
        elif self.allAgentsInHell_after:
            return normalCost + self.hellDict[sPrime]
        else:
            if s == sPrime:
                return normalCost
            elif sPrime in self.priestDict.keys():
                return normalCost + self.priestDict[sPrime]
            else:
                return normalCost + middleStageDict[sPrime]
```

Observation Function (All Agents)

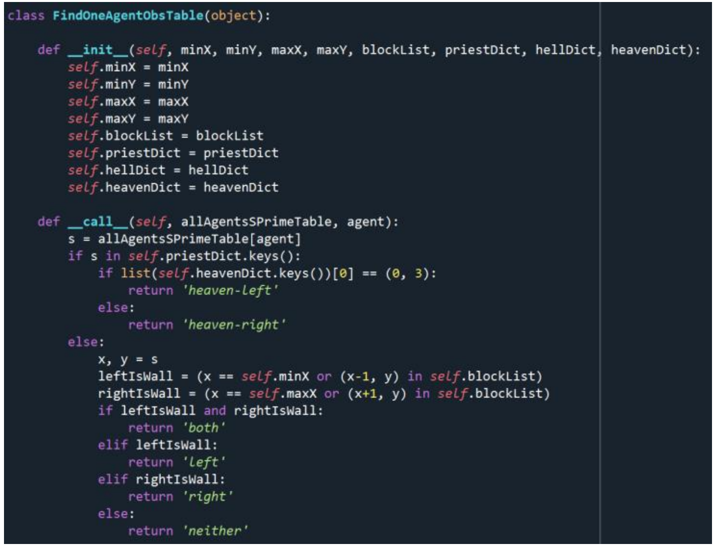
- $\mathrm{p}(o|s,a)$ where o, s, a are vectors representing the, observations original states, and actions of all agents
- For this project, ignore the communication observations (future improvement)



```
class ObservationFunction(object):

    def __init__(self, findOneAgentObsTable):
        self.findOneAgentObsTable = findOneAgentObsTable

    def __call__ (self, allAgentsSPrimeTable, allAgentsActionTable, allAgentsObsTable):
        allAgentsObsExpected = {agent: self.findOneAgentObsTable(allAgentsSPrimeTable, agent) for agent in allAgentsActionTable.keys()}
        return int(allAgentsObsTable == allAgentsObsExpected)
```

Observation Function (One Agent)

- $\mathrm{p}(o|s, a)$ where o, s, a are vectors representing the, observations original states, and actions of all agents
- For this project, ignore the communication observations (future improvement)



```python
class FindOneAgentObsTable(object):

    def __init__(self, minX, minY, maxX, maxY, blockList, priestDict, hellDict, heavenDict):
        self.minX = minX
        self.minY = minY
        self.maxX = maxX
        self.maxY = maxY
        self.blockList = blockList
        self.priestDict = priestDict
        self.hellDict = hellDict
        self.heavenDict = heavenDict

    def __call__(self, allAgentsSPrimeTable, agent):
        s = allAgentsSPrimeTable[agent]
        if s in self.priestDict.keys():
            if list(self.heavenDict.keys())[0] == (0, 3):
                return 'heaven-left'
            else:
                return 'heaven-right'
        else:
            x, y = s
            leftIsWall = (x == self.minX or (x-1, y) in self.blockList)
            rightIsWall = (x == self.maxX or (x+1, y) in self.blockList)
            if leftIsWall and rightIsWall:
                return 'both'
            elif leftIsWall:
                return 'left'
            elif rightIsWall:
                return 'right'
            else:
                return 'neither'
```

Future Improvements

- Implement the communication component
- Add choices of observation function. Currently, to line up with set up in *Decentralized Planning under Uncertainty for Teams of Communicating Agents (Spaan et al.)*, the environment uses “heaven-left” and “heaven-right” to represent the location of Heaven and Hell. In scenarios of multiple Heavens and Hells, the functions need to be adjusted to show the correct outputs.
- Consider scenarios when users not giving the correct input. Throw Warnings or Errors

Thank You