

**VIETNAM NATIONAL UNIVERSITY - HO CHI
MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE & ENGINEERING**



INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Course by Dr. Nguyen Trung Ky

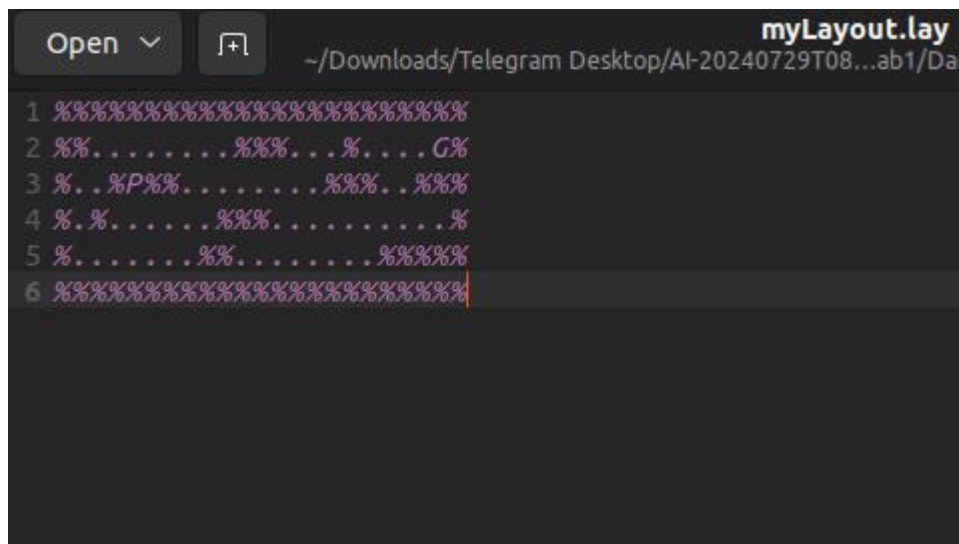
LAB #1: DESIGNING PAC-MAN AGENTS

Student's Name: Nguyễn Đức Nguyên Phúc
Student's ID: ITDSIU21108

Exercise 1: It is random, thus in the end, it will always get the food, even if it takes some time to do so. It is not going to collapse because it solely seeks legal action. The agent travels quickly in a variety of directions; it can move to a new location or loop back to the previous step. If there are adversaries, there's a strong chance of losing because the agent can't run away from danger, and it might take some time for an agent to reach the food position, so the outcome isn't always ideal. RandomAgent travels randomly, even when it stops. On occasion, it stops on the wall.

Exercise 2:

myLayout.lay:

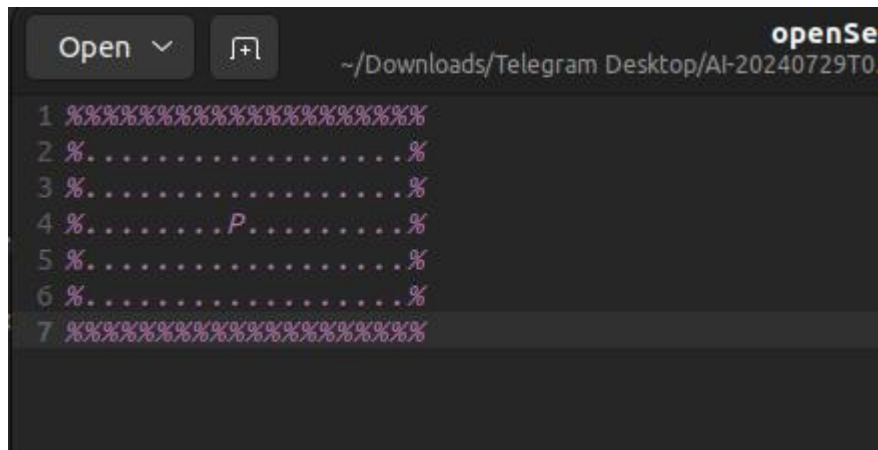


```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%
2 %%. . . . . %%. . . % . . . . G%
3 %. . %P%. . . . . %%. . . %%. . . %%. . . %
4 %. % . . . . . %%. . . . . %
5 %. . . . . %%. . . . . %%. . . . %
6 %%%%%%%%%%%%%%%%%%%%%%%%%%
```

Game interface:

- For the myLayout.lay, each "%" defines the pane or wall for the game rendering; these can take on a variety of forms, but "%" must wrap around the map as a precondition.
- This means that the game system can function flawlessly and error-free.
- The foods that the agent can consume to win the game are defined by each ".".
- The map will depict the layout as a place that the agent can navigate over if it contains no ".".
- The agent's (Pac-man) starting position is indicated by the "P".
- The enemies' starting positions are indicated by the "G".

openSearch.py



First try: 200 in 11.07 seconds

Second try: 115 in 11.0 seconds

Third try: 160 in 11.01 seconds

Try 4: 210 in 10.97 seconds

Try 5 - 99 in 11.10 seconds

Try 6: 190 in 11.03 seconds

=> With RandomAgent in the openSearch environment, average 162.3 points in 11s.

It is less likely to eat food on the edges and usually eats everything in the center.

Exercise 3:

BetterRandomAgent class:

```
class BetterRandomAgent(Agent):
    def getAction(self, state):
        legalActions = state.getLegalPacmanActions()
        legalActions.pop()
        for legalAction in legalActions:
            successorState = state.generatePacmanSuccessor(legalAction)
            if successorState.getNumFood() < state.getNumFood():
                return legalAction
        return random.choice(legalActions)
```

First try: 225 in 11.07 seconds
Try #2: 212 in 11.15 seconds
Try 3: 160 in 11.0 seconds
Try 4: 256 in 11.01 seconds
Try 5: 145 in 11.12 seconds
Try 6: 155 in 11.10s

=> With BetterRandomAgent in the openSearch environment, an average of 192.17 points in 11s was achieved.

BetterRandomAgent follows a random path without pausing. BetterRandomAgent moves faster than RandomAgent. It is penalized with less points as a result. The agent travels along the food path; if there are gaps, it will travel at random until it locates the food in the openSearch arrangement.

Exercise 4:

ReflexAgent class:

```
class ReflexAgent(Agent):
    def getAction(self, state):
        legalActions = state.getLegalPacmanActions()
        legalActions.pop()
        for legalAction in legalActions:
            successorState = state.generatePacmanSuccessor(legalAction)
            if successorState.getNumFood() < state.getNumFood():
                return legalAction
        return random.choice(legalActions)
```

In an openSearch environment, the agent always consumes the full right side of the map using the zic-zac approach. It will then randomly venture to the opposite side of the map in search of food pellets, which it will thereafter devour in their entirety. Naturally, the remaining portion keeps using zic-zac to eat all of the food. When utilizing the ReflexAgent class in the openSearch layout, the game takes a lot longer to finish than when using the BetterRandomAgent layout. On the other hand, there are certain similarities between them.

Pac-man's position: **gameState.getPacmanPosition()**

All ghosts' position: **gameState.getGhostPositions()**

Walls' location: **gameState.getWalls()**

Capsules' position: **gameState.getCapsules()**

Each food pellet's position: `gameState.getFood()`

The total number of food pellets still available: `gameState.getNumFood()`

Whether it has won or lost the game: `gameState.isLose()`, `gameState.isWin()`

Pac-man's current score in the game: `gameState.getScore()`

```
def getPacmanPosition(self):
    return self.data.agentStates[0].getPosition()

def getGhostStates(self):
    return self.data.agentStates[1:]

def getGhostState(self, agentIndex):
    if agentIndex == 0 or agentIndex >= self.getNumAgents():
        raise Exception("Invalid index passed to getGhostState")
    return self.data.agentStates[agentIndex]

def getGhostPosition(self, agentIndex):
    if agentIndex == 0:
        raise Exception("Pacman's index passed to getGhostPosition")
    return self.data.agentStates[agentIndex].getPosition()

def getGhostPositions(self):
    return [s.getPosition() for s in self.getGhostStates()]
```

```
def getNumFood(self):
    return self.data.food.count()

def getFood(self):
    """
    Returns a Grid of boolean food indicator variables.

    Grids can be accessed via list notation, so to check
    if there is food at (x,y), just call

    currentFood = state.getFood()
    if currentFood[x][y] == True: ...
    """
    return self.data.food
```

```
def isLose(self):
    return self.data._lose

def isWin(self):
    return self.data._win

#####
#           Helper methods:           #
# You shouldn't need to call these directly #
#####
```