**Student Name: Phan Tran Thanh Huy  
Student ID: ITCSIU22056**

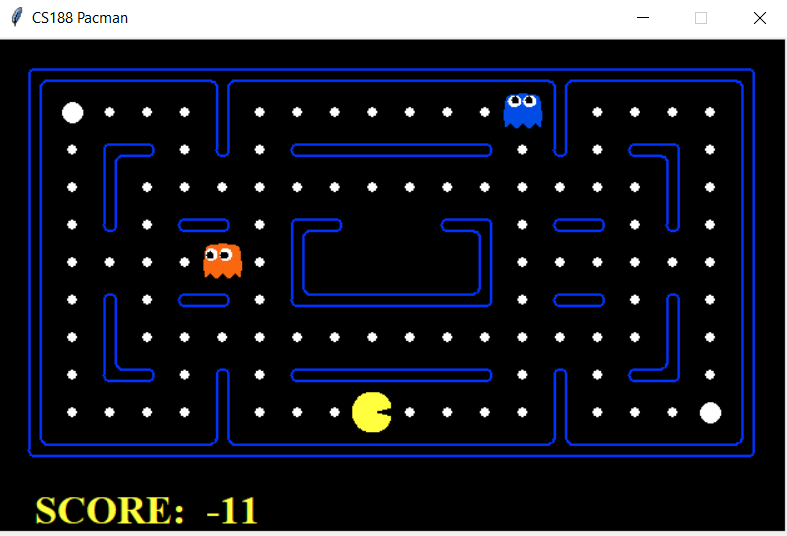
**IT159IU: Artificial Intelligence**

**Lab#1/Assignment#1: Designing Pac-Man Agents**

# Introduction

In this lab assignment, you will familiarize yourself with the Pac-Man World. Over the next few assignments, you will implement your Pac-Man agent to find paths through its maze world in order to reach a particular location and to collect food efficiently. You will also build general search algorithms and apply them to Pac-Man scenarios.

The code for the assignment consists of several Python files, some of which you will need to read and understand in order to complete assignments, and some of which you can ignore. The Pac-Man code was developed by John DeNero and Dan Klein at UC Berkeley for the class CS188. The code is written in Python 3.x, which you require an interpreter for in order to run the lab assignment.



*Figure 1: The Pac-Man World*

# Step 1: Download Code

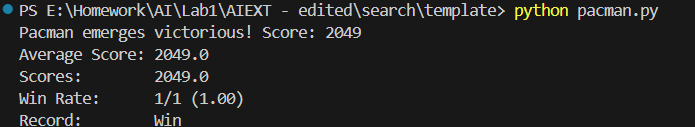
The code you will be using can be downloaded as a zip archive on Blackboard, namely **search**.

Extract the files into a directory/folder on your computer. A folder called, search will be created and in it you will find several dozen files. To ensure that you have a working version of the files, run the following command:

python pacman.py

You should see a game screen pop up (see Figure 1). This is a basic Pac-Man game. In the game, you control the movements of Pac-Man using arrow keys on your keyboard. Go ahead and try it.

The Pac-Man world is laid out as corridors (with shiny blue walls) where Pac-Man can move about. Little white pellets are sometimes littered throughout the corridors. This is food for Pac-Man (larger pellets are power food or capsules, try and figure out what those are for). In the world shown in Figure 1, PacMan has adversaries: colored ghosts that eat Pac-Man when it runs into them. Ghosts move about without eating any food. When Pac-Man is eaten, it dies and the game ends. The screen will disappear.



# Step 2: Pac-Man Agent

In this and the next few assignments, you will be writing agent programs to control the actions of PacMan. That is, creating a Pac-Man agent. The code enables you to use different environments to try out your Pac-Man agent programs. To specify an environment (for example, **testMaze**), you use the command:

python pacman.py -–layout testMaze



Ảnh có chứa văn bản, Phông chữ, phần mềm, ảnh chụp màn hình

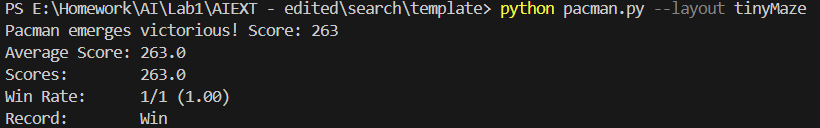
Mô tả được tạo tự động

Go ahead and try it. It is a simple maze with one corridor. Here is one you will use more often:

python pacman.py -–layout tinyMaze

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, biểu đồ

Mô tả được tạo tự động



*Figure 2: Pac-Man agent in tinyMaze.*

There are several other environments defined: **mediumMaze**, **bigMaze, openSearch**, etc. You can also vary the scale of the screen by using the **–zoom** option as shown below:

python pacman.py –-layout tinyMaze –-zoom 2

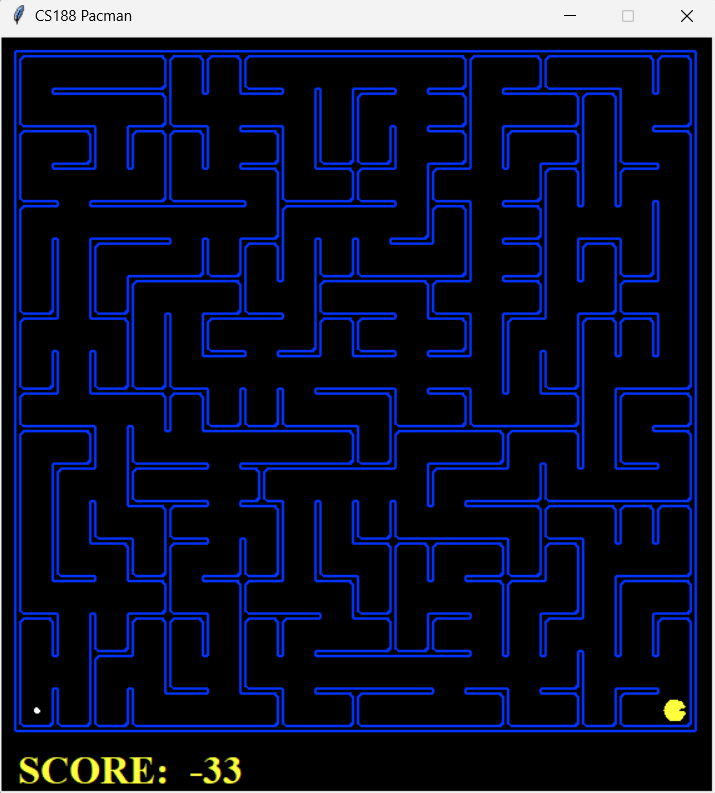
Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

python pacman.py –-layout bigMaze –-zoom 0.5



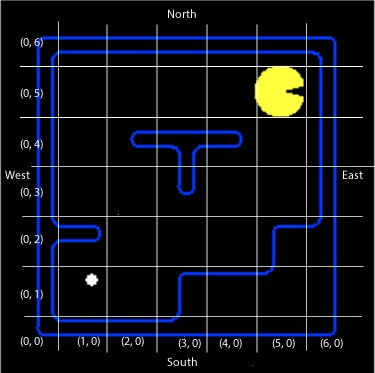
Ảnh có chứa văn bản, phần mềm, Phông chữ, Phần mềm đa phương tiện

Mô tả được tạo tự động

All of these are single agent environments, the agent being Pac-Man. In these environments, Pan-Man always starts at the top right corner and, at the bottom left corner is a single food pellet (see picture above). The game ends when Pac-Man eats very last pellet (there can be pellets anywhere in its world).

# Step 3: Learning the Pac-Man Grid and Actions

**Grid:** The environment is essentially a grid of squares. At any given time, Pac-Man occupies a square and faces one of the four directions: North, South, East, or West. There may be walls in between the square (like the t-shaped wall in **tinyMaze**) or entire squares might be blocked by walls (like the bottom right corner of **tinyMaze**. regardless, the location of Pac-Man is determined by the x- and y- coordinates of the grid (as shown below):



*Figure 3: The Pac-Man Grid. Pac-Man is at position (5, 5). Food pellet is at (1, 1)*

**Actions**

Pac-Man can only carry out the following actions:

* ‘North’: go one step north
* ‘South’: go one step south
* ‘East’: go one step east
* ‘West’: go one step west
* ‘Stop’: stop, do not move

Below, you will see how these are specified to be carried out.

# Step 4: Diving into Some Code

Now that you are familiar with the basic world, it is time to get familiar with some of the code. Start by looking at the contents of the file **game.py**.

Skim through the parts worth reading section of the code. Focus first on the following classes: **Agent**, **Directions**, and **Configuration**.

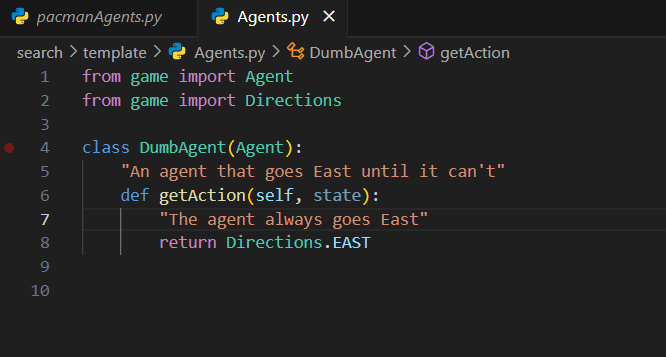
**Agent**

The **Agent** class is very simple. It is the class you will subclass to create your Pac-Man agent. For example, here is a very simple, and dumb, agent:

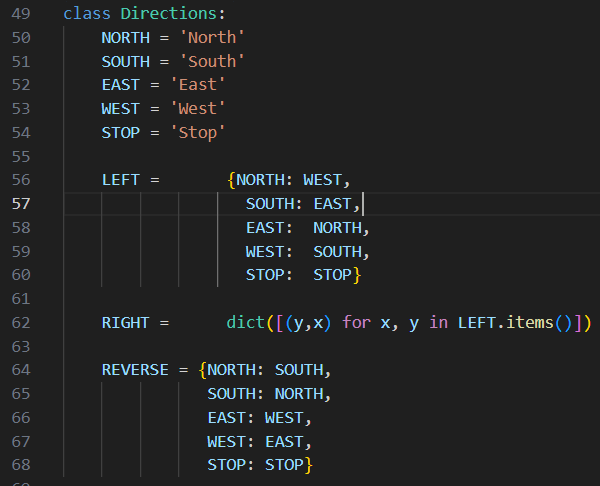
**from** game **import** Agent  
**from** game **import** Directions  
  
**class** DumbAgent(Agent):  
 *"An agent that goes East until it can't"* **def** getAction(self, state):  
 *"The agent always goes East"* **return** Directions.EAST

The way it is set up, when you specify to the game (see below) that the Pac-Man will be controlled by an instance of a **DumbAgent**, the action returned by the **getAction()** method will be carried out at each time step. Important things to note in the above code are:

* You should create a new file called, **Agents.py**, in the same directory/folder as the rest of the code base. Enter the code above exactly as shown. Be sure to save the file.



* Every subclass of **Agent** (like **DumbAgent**) is required to implement a **getAction()** method. This is the method called in each time step of the game and as mentioned above, it should return a valid action for Pac-Man to carry out.
* Notice that we are importing the classes **Agent** and **Directions** from **game.py**.
* The **getAction()** method is supplied a parameter: **state**, which it can use to find out about the current game state (more on this below). For now, we are ignoring it.
* Study the class **Directions** (defined in **game.py**).

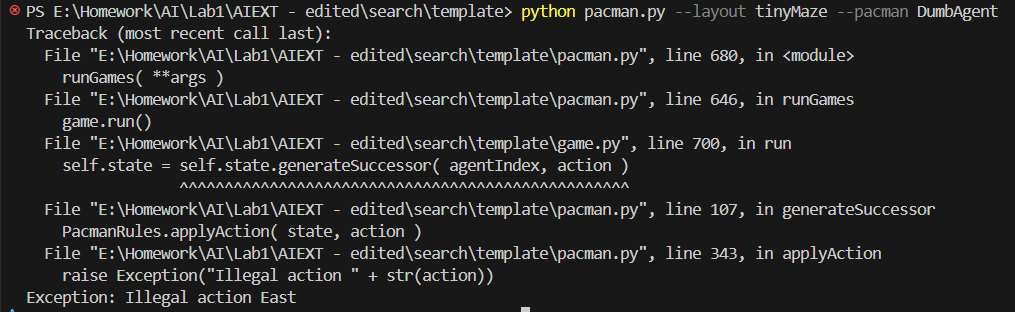


# Step 5: Run the code

Next run the Pac-Man game with its control as **DumbAgent** using the command:

python pacman.py –-layout tinyMaze –-pacman DumbAgent

The command above is specifying to run the Pac-Man game using the **tinyMaze** environment and the agent is controlled by the **DumbAgent**. What happens? It returns an exception.



In the Pac-Man game, if the path to the grid is blocked and Pac-Man tries to go into it, the game crashes with an “**Illegal action**” exception. This is OK. After all, it is a dumb agent. We’ll fix that next. Try the same agent in the **mediumMaze**. Same result, right? Good!

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, phần mềm

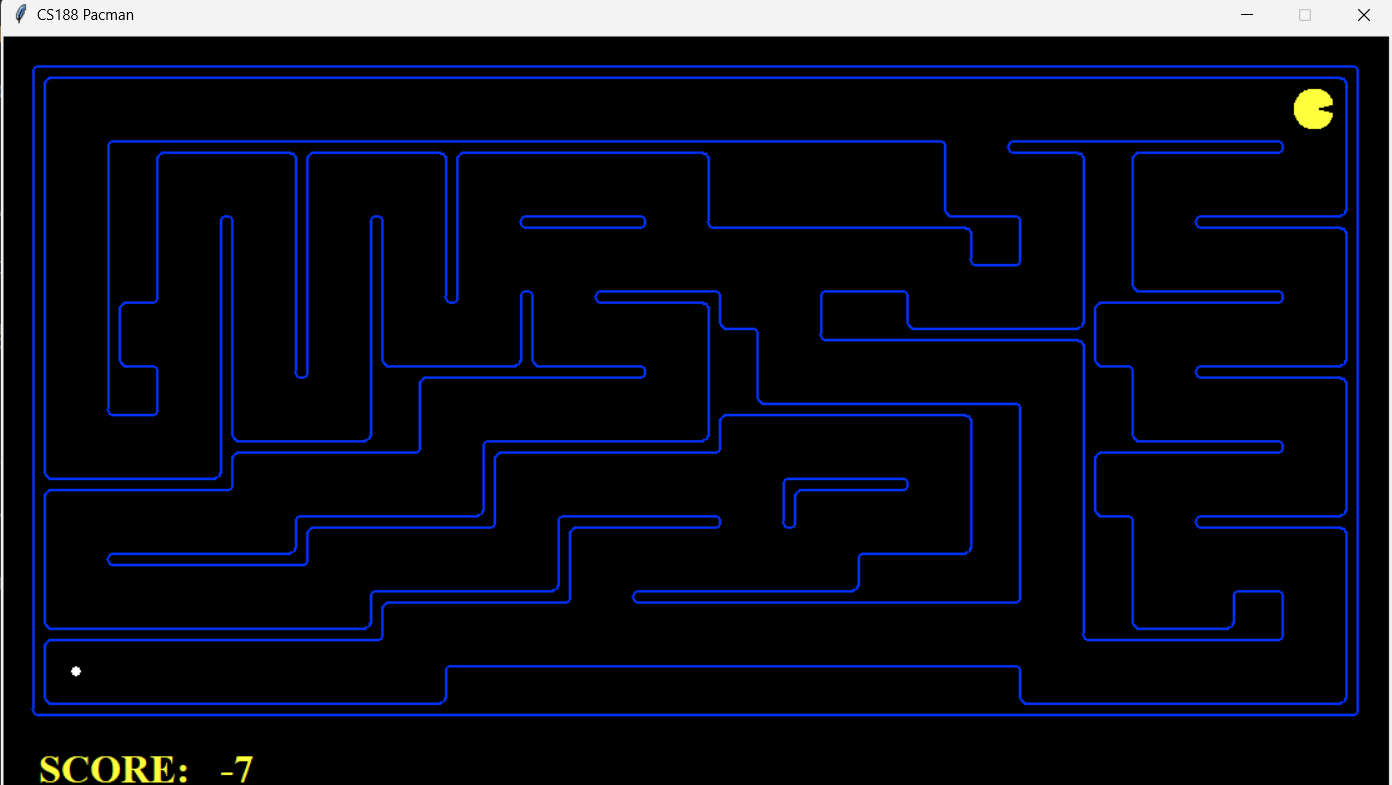
Mô tả được tạo tự động

# Step 6: Learning about GameState

Next, lets us try and use the information present in the state parameter. This is an object of type **GameState** which is defined in the file **pacman.py**. Study the **GameState** class closely and note the methods defined. Using these, you can get all kinds of information about the current state of the game. Then you can base your agent’s action accordingly. Below, we show how you can use some of these and prevent the game from crashing.

**class** DumbAgent(Agent):  
 *"An agent that goes East until it can't."* **def** getAction(self, state):  
 *"The agent receives a GameState (defined in pacman.py)."* print(**"Location: "**, state.getPacmanPosition())  
 print(**"Actions available: "**, state.getLegalPacmanActions())  
 **if** Directions.EAST **in** state.getLegalPacmanActions():  
 print(**"Going East."**)  
 **return** Directions.EAST  
 **else**:  
 print(**"Stopping."**)  
 **return** Directions.STOP

As in Step 4, save this version of your program in **Agents.py** and run it on **tinyMaze**, as well as **mediumMaze**. Observe the behavior. Try out some of the other methods defined in **GameState** to get an idea of what information is available to your agent.



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

# Step 7: A Random Agent

OK, now it is time to write your own agent code.

**Exercise 1:** Create a new class called, **RandomAgent** (in the **Agents.py** file), which based on the current options and pick a random action to carry out. Run your agent in the **tinyMaze** environment as well as **mediumMaze** environment. Observe the agent’s behavior. Does it get to the food? Always? Without crashing? Etc.



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

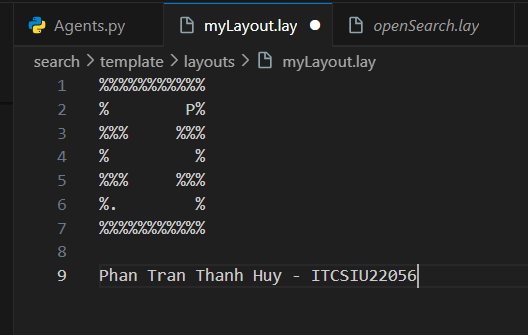
Mô tả được tạo tự động

**Describe:** For each action, the method getLegalPacmanActions() method will return all the possible actions that the pacman can move. Then the legal\_actions will be declared and save all the possible actions. After that, It will check if there are legal\_actions and return the action. Otherwise, it will stop and generate a new action.

# Step 8: Exploring Environments

See the files in the folder/directory **layouts**. Environments are specified using simple text files (**\*.lay**) which are then rendered nicely by the graphics modules in the code base. Examine several layout files to see how to specify walls, ghosts, pacman, food, etc.

**Exercise 2:** Create a small environment of your own. Make sure it has walls and corridors, as well as some food. Save it as **myLayout.lay** in the **layouts** directory.



Run your **RandomAgent** in this environment and observe how it does.

Ảnh có chứa văn bản, ảnh chụp màn hình, biểu đồ

Mô tả được tạo tự động

Ảnh có chứa văn bản, Phông chữ, ảnh chụp màn hình

Mô tả được tạo tự động

Also, try your agent out in the **openSearch** environment (files are already provided in the layouts directory). Run your agent several times and record, on average, what score you get.

Ảnh có chứa ảnh chụp màn hình, văn bản, Hình chữ nhật

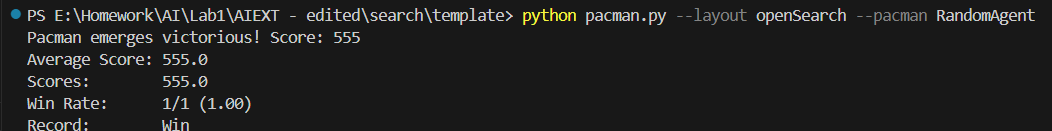
Mô tả được tạo tự động

* 1st Attemp:

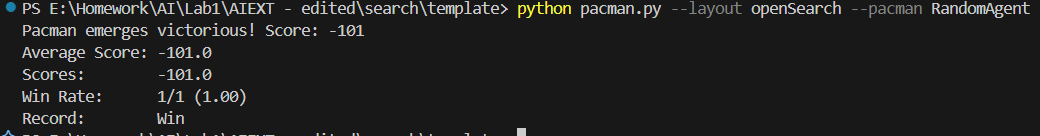
Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Phông chữ

Mô tả được tạo tự động

* 2nd Attemp:



* 3rd Attemp:



# Step 9: A Better Random Agent

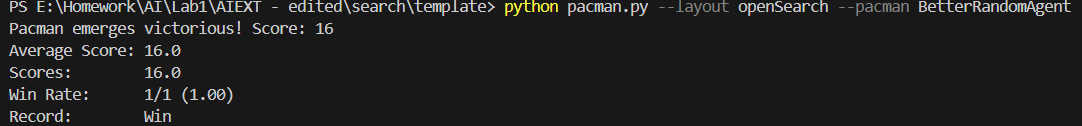
If you print out and look at the choice of actions at each step, you will notice that **RandomAgent** always includes a choice for the ‘Stop’ action. This tends to slow it down. Stopping is needed in situations where you need to evade ghosts. For now, in environments without any ghosts, you can choose not to pick the ‘Stop’ action.

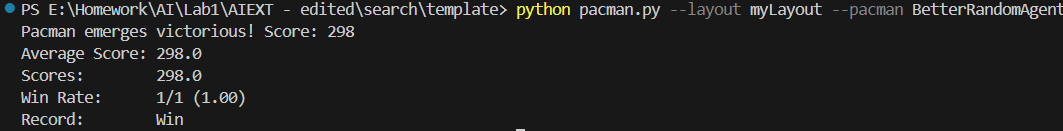
**Exercise 3:** Create a new class called **BetterRandomAgent** (in the file **Agent.py**) so that it never chooses ‘Stop’ as its action. Run the agent in **openSearch** and **myLayout** environments and observe how it does.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

**Describe:** Now, RandomAgent() will avoid choosing 'Stop' unless it's the only available action. This should make it move more efficiently in environments without ghosts.





# Step 10: Percepts

What the Pac-Man agent can perceive is based on the methods of the **GameState** class which is defined in the file **pacman.py**. Open this file and let's look through the options.

It is important to realize that the game has several different agents (Pac-Man and the ghosts). Each agent in the game has a unique **index**; Pac-Man is always **index** 0, with ghosts starting at **index** 1.

Pac-Man can perceive:

* His position
* The position of all the ghosts
* The locations of the walls
* The positions of the capsules
* The positions of each food pellet
* The total number of food pellets still available
* Whether it has won or lost the game
* His current score in the game

In addition, Pac-Man can also determine given the action it chooses what the next state of the environment will be, by using the method **generatePacmanSuccessor()**. It is clear from the methods available here that Pac-Man's environment is fully observable. Pac-Man's environment is also static because until it decides what to do and takes an action, the ghosts do not move.

**Exercise 4:**In the file **Agents.py** create a new agent called **ReflexAgent**. This agent should look at the possible legal actions, and if one of these actions would cause a food pellet to be eaten, it should choose that action. If none of the immediate actions lead to food, it should choose randomly from the possibilities (excluding 'Stop'). Test your agent in both the **openSearch** and **myLayout** layouts.

python pacman.py --l openSearch --p ReflexAgent

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, Phông chữ, phần mềm, ảnh chụp màn hình

Mô tả được tạo tự động

**Describe:** The agent gets a list of all valid actions Pac-Man can take in the current state using state.getLegalPacmanActions() method. Then it will have a condition that if STOP is an option and there are other available moves, it removes STOP to avoid unnecessary pauses. The agent will simulate each possible move by calling state.generatePacmanSuccessor(action) method, which returns the game state if Pac-Man takes that action. If the successor state has fewer food pellets than the current state, it means Pac-Man will eat food, so the agent immediately chooses that action. If none of the legal actions lead to eating food, the agent picks a random action from the remaining legal moves.

**Percepts**

* His position: **gameState.getPacmanPosition()**
* The position of all the ghosts: **gameState.getGhostPosition(agentIndex)**
* The locations of the walls: 
* The positions of the capsules: Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

  Mô tả được tạo tự động
* The positions of each food pellet: Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, đồng hồ

  Mô tả được tạo tự động
* The total number of food pellets still available: 

Ảnh có chứa ảnh chụp màn hình, văn bản, Phông chữ

Mô tả được tạo tự động

* Whether it has won or lost the game:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động