|  |  |  |  |
| --- | --- | --- | --- |
|  | Logo  Description automatically generated | | **Designing Pac-Man Agents.** |
| Lab #1 |
|  | **laboratory** |  | **Nguyen Thi Mai Huong – ITITIU19128**  **Part 1.a**: Behavior of **RandomAgent** from Step 7   * The class RandomAgent is filed Agents.py firstly gets the legal moveable actions then randomly picks a legal direction to directly coordinate the agent to that action. At each time, the system prints out a list of legal actions and the random action that the agent will go in the next step. * Observer: since the agent’s direction is randomly picked, its behavior is spontaneous without any optimization in picking directions. The agent can get to the food if luckily after dozens of movement steps including many times temporary stop. * Source code:     **Part 1.b**: The screenshot of my **myLayout** environment from Step 8    **Part 1.c**: Behavior of **BetterRandomAgent** from Step 9   * This random agent class virtually is implemented similarly to the **RandomAgent** class from Step 7, however, it removes ‘*the stop’* from the legal list before picking any legal direction in this list.     **Part 1.d + 1.e**: Behavior of **ReflexAgent** from Step 10   * The reflect agent class will get the legal action with the maximum score agent can get (this means the chosen direction must have food or something to make a greater score. It performs the following steps:   + Print Pac man’s position:  print("Current location of Pacman: ", state.getPacmanPosition())  + List all possible directions that the agent can move:  legalActions = state.getLegalPacmanActions()          print ("Legal moves: ", legalActions)  + If the movable directions list contains a stop action then remove it:  if 'Stop' in legalActions: legalActions.remove('Stop')  + Evaluate the score of each legal action that the agent can pick and pick the directions with max score. In the case that all actions return the same score 🡪 pick randomly a legal action in the list.  scores = [self.evaluationAction(state, action) for action in legalActions]          bestscore = max(scores)          print("Scores : ", scores)          bestIndices = [index for index in range(len(scores)) if scores[index] == bestscore]          chosenIndex = random.choice(bestIndices)          print("Go", legalActions[chosenIndex])          return legalActions[chosenIndex]      def evaluationAction(self, state, action):          successorGameState = state.generatePacmanSuccessor(action)          return successorGameState.getScore() |
|  |  |  | * The demonstration: |