# Islamic University of Technology


---

# Report on Lab 04

---

(CSE 4618 Artificial Intelligence Lab)


| | |
|---|---|
| Name: | **Md Mahmudul Islam Mahin** |
| Student ID: | **210042140** |
| Department: | CSE |
| Program: | BSc in SWE |
| Date: | $3^{rd}$ September 2025 |

# Contents


**1 Introduction** **2**

**2 Question 01: Reflex Agent** **2**

**3 Question 02: Minimax** **2**

3.1 getAction(self, gameState): . . . . . 2

3.2 value(self, gameState, agentIndex, depth): . . . . . 2

3.3 minimizer(self, gameState, agentIndex, depth): . . . . . 3

3.4 maximizer(self, gameState, agentIndex, depth): . . . . . 3

**4 Question 03: Alpha-Beta Pruning** **3**

4.1 getAction(self, gameState): . . . . . 3

4.2 value(self, gameState, agentIndex, depth, alpha, beta): . . . . . 4

4.3 minimizer(self, gameState, agentIndex, depth, alpha, beta): . . . . . 4

4.4 maximizer(self, gameState, agentIndex, depth, alpha, beta): . . . . . 4

**5 Question 04: Expectimax** **5**

5.1 value(self, gameState, agentIndex, depth): . . . . . 5

5.2 expectation (self, gameState, agentIndex, depth): . . . . . 5

**6 Question 05: Evaluation Function** **5**

**7 GitHub Repository** **6**

# 1 Introduction

In this lab, we will design agents for the classic version of Pacman, including ghosts. Along the way, we will implement both minimax and expectimax search and try our hand at evaluation function design.

# 2 Question 01: Reflex Agent

In order to design a suitable evaluation function for our reflex agent, we have to define the evaluationFunction(self, currentGameState, action) in such a way that the good features are rewarded and the bad features are punished. One way to do it is to set the score as the weighted value of some features of the currentGameState and SuccessorGameState. For my solution the features that I've chosen are closestFoodDistance, closestGhostDistance and RemainingFoods. And the corresponding weight factors were chosen based on intuition, proper logic and some trial and error. Initially we define two lists to store foodDistances and ghostDistances from the current pacman position. Both were initialized with 1e9 to avoid issues if there are no remaining foods or the pacman and the ghost are in the same position.
A score variable was declared and set to 0. Then totalFoods was determined taking the length of the foods in currentGameState. RemainingFood was obtained by subtracting the number (length) of foods in the successorGameState. We calculated and added the Manhattan Distance from the pacman's position to each food item into the foodDistances list, and similarly, we computed the distances from pacman to the ghosts and stored them in the GhostDistances list. From the foodDistances and ghostDistances list we obtain the minimum values and assign them as closestFoodDistance and closestGhostDistance.
We try the reflex agent on the default mediumClassic layout and observe the behavior of our reflex agent for both the user defined evaluator function as well as for successorGameState.getScore() as evaluator. When we use successorGameState.getScore() as the evaluator, our agent often moves back and forth and ends up losing but with the user defined evaluation function the agent performs better and wins in all cases.

# 3 Question 02: Minimax

To write an adversarial minimax search agent we have to define some functions –

## 3.1 getAction(self, gameState):

The value(gameState, agentIndex, depth) returns a tuple consisting of the score and the nextAction, since we are only interested in the next action we will return the first Index (action) of the function's output. As the pacman is placed on the top of the graph, the AgentIndex will be 0 denoting the pacman.

## 3.2 value(self, gameState, agentIndex, depth):

The value function returns a tuple, a score (of an evaluation function) and the next action. The output of the function depends on the following conditions –

**Terminal Node:**

If the node is a terminal node or the deepest node the only choices are that it's a Lose or Win state. When the node is a terminal node, it returns the result of the evaluation function of the

GameState as output along with STOP as the next action.

**Pacman (agentIndex = 0):**

If the current node is a Pacman having agent index 0, then it will call the maximizer(gameState, agentIndex, depth) function. As pacman always wants to maximize the score by playing optimally, it will use the maximizer function.

**Ghosts (agentIndex = 1 'N):**

If the current node is a Ghost having non-zero agent index , then it will call the minimizer(gameState, agentIndex, depth) function. As they aim to minimize the score or utility for the player (Max) the function calls the minimizer function for ghosts.

### 3.3 minimizer(self, gameState, agentIndex, depth):

The minimizer function takes the gameState, agentIndex (ghost or greater than 0 in this case) and depth of the game (ply) as input and returns the currentScore and currentAction as a tuple. At first it determines the index of the next agent by modding it with the total number of agents. If the nextAgent is 0 (pacman) it means a ply has been completed so the depth is reduced, otherwise it remains as it was. Then we obtain the actionList from the available LegalActions for the nextAgent, iterate over actionList and obtain successorGameState and successorScore using the value function (index 0 denoting the score) and return the minimum score and the action leading to it as output.

### 3.4 maximizer(self, gameState, agentIndex, depth):

The maximizer function takes the gameState, agentIndex (pacman or 0 in this case) and depth of the game (ply) as input and returns the currentScore and currentAction as a tuple. At first it determines the index of the next agent by modding it with the total number of agents. If the nextAgent is 0 (pacman) it means a ply has been completed so the depth is reduced, otherwise it remains as it was. Then we obtain the actionList from the available LegalActions for the nextAgent, iterate over actionList and obtains successor's GameState and successorScore using the value function (index 0 denoting the score) and returns the maximum score and the action leading to it as output.

## 4 Question 03: Alpha-Beta Pruning

The solution is quite similar to that of the minimax problem but here we introduce two new parameters, Alpha and Beta.

### 4.1 getAction(self, gameState):

For alpha-beta pruning we introduce two more parameters to the value function – alpha and beta. The value(gameState, agentIndex, depth, alpha, beta) returns a tuple consisting of the score and the nextAction, since we are only interested in the next action we will return the first Index (action) of the function's output. As the pacman is placed on the top of the graph, the AgentIndex will be 0 denoting the pacman. The values of Alpha and Beta at the top are initially assigned as a very small (–1e9) and very large (1e9) number respectively as our goal is to maximize alpha and minimize beta

## 4.2 value(self, gameState, agentIndex, depth, alpha, beta):

The value function takes two more parameters as input (alpha and beta) and returns a tuple, a score (of an evaluation function) and the next action. The output of the function depends on the following conditions –

**Terminal Node :**

If the node is a terminal node or the deepest node the only choices are that it's a Lose or Win state. When the node is a terminal node, it returns the result of the evaluation function of the GameState as output along with STOP as the next action.

**Pacman (agentIndex = 0) :**

If the current node is a Pacman having agent index 0, then it will call the maximizer(gameState, agentIndex, depth) function. As pacman always wants to maximize the score by playing optimally, it will use the maximizer function.

**Ghosts (agentIndex = 1  N) :**

If the current node is a Ghost having non-zero agent index , then it will call the minimizer(gameState, agentIndex, depth) function. As they aim to minimize the score or utility for the player (Max) the function calls the minimizer function for ghosts.

## 4.3 minimizer(self, gameState, agentIndex, depth, alpha, beta):

The minimizer function takes the gameState, agentIndex (ghost or greater than 0 in this case), depth (ply), alpha and beta value of the game as input and returns the currentScore and currentAction as a tuple. At first it determines the index of the next agent by modding it with the total number of agents. If the nextAgent is 0 (pacman) it means a ply has been completed so the depth is reduced, otherwise it remains as it was. Then we obtain the actionList from the available LegalActions for the nextAgent, iterate over actionList and obtain successorGameState and successorScore using the value function (index 0 denoting the score) and acquire the minimum score as currentScore and the action leading to it as currentAction. If the currentScore is lower than alpha value it returns the minimum score and the action leading to it as output as the value of alpha cant be improved or maximized any further thus we don't need to explore the current branch so we prune it by returning the result and not exploring any further (stopping the loop). But if the current score is greater than that of Alpha, we update the Beta to the minimum of the currentScore and the existing Beta and return the currentScore and currentAction after completing the whole iteration.

## 4.4 maximizer(self, gameState, agentIndex, depth, alpha, beta):

The maximizer function takes the gameState, agentIndex (ghost or greater than 0 in this case), depth (ply), alpha and beta value of the game as input and returns the currentScore and currentAction as a tuple. At first it determines the index of the next agent by modding it with the total number of agents. If the nextAgent is 0 (pacman) it means a ply has been completed so the depth is reduced, otherwise it remains as it was. Then we obtain the actionList from the available LegalActions for the nextAgent, iterate over actionList and obtain successorGameState and successorScore using the value function (index 0 denoting the score) and acquire the maximum score as currentScore and the action leading to it as currentAction. If the current score is higher than beta value it returns the maximum score and the action leading to it as

output as the value of beta can't be improved or minimized any further thus we don't need to explore the current branch so we prune it by returning the result and not exploring any further (stopping the loop). But if the current score is smaller than that of Beta, we update the Alpha to the maximum of the currentScore and the existing Alpha and return the currentScore and currentAction after completing the whole iteration.

# 5 Question 04: Expectimax

The solution approach is quite similar to that of minimax but the minimizer function is replaced with expectation function. Here only the changes have been discussed,

## 5.1 value(self, gameState, agentIndex, depth):

The output of the value function depends on the following conditions –

**Terminal Node :**

If the node is a terminal node or the deepest node the only choices are that it's a Lose or Win state. When the node is a terminal node, it returns the result of the evaluation function of the GameState as output along with STOP as the next action.

**Pacman (agentIndex = 0) :**

If the current node is a Pacman having agent index 0, then it will call the maximizer(gameState, agentIndex, depth) function. As pacman always wants to maximize the score by playing optimally, it will use the maximizer function.

**Ghosts (agentIndex = 1 N) :**

If the current node is a Ghost having non-zero agent index , then it will call the expectation(gameState, agentIndex, depth) function. As they aim to minimize the expected utility for the player (Max) the function calls the expectation function for ghosts.

## 5.2 expectation (self, gameState, agentIndex, depth):

The expectation function takes the gameState, agentIndex (ghost or greater than 0 in this case) and depth of the game (ply) as input and returns the currentScore and currentAction as a tuple. At first it determines the index of the next agent by modding it with the total number of agents. If the nextAgent is 0 (pacman) it means a ply has been completed so the depth is reduced, otherwise it remains as it was. Then we obtain the actionList from the available LegalActions for the nextAgent and initialize two variables, the $\text{expected}_v alue as 0 and probability value as 1 by length of action List assuming all the actions are equally likely.$
$After that, we iterate over action List and obtain successor GameState and successor Score using the value func$

# 6 Question 05: Evaluation Function

For designing a better evaluation function we need an evaluation score which is the weighted average of some features. So, to design a better evaluation function we at first obtained the pacmanPosition and ghostPositions from the currentGameState. Then we acquire the foodList as a list from the currentGameState and get the foodCount and capsuleCount as

length of foodList and currentGameState.getCapsules() respectively. We also declare a variable named closestFood as 1 and obtain the gameScore from currentGameState.getScore(). Then we calculate the Manhattan distances between Pac-Man's current position (pacmanPosition) and the positions of all the food items in the foodList. These distances are stored in the $food_distanceslist. If there's at least one food item (foodCount > 0) we find the closest food item by using them i$

# 7 GitHub Repository

I will be uploading the lab tasks in the following repository: CSE 4618: Artificial Intelligence