QUESTION 1:

a) Key concepts:

- An agent is anything that carries out an activity in an environment using actuators in response to signals autonomously picked up using sensors.

- An agent function is a mathematical function that governs the actions of an agent. It can also be referred to as the core logic of an agent; That is, it governs how the agents use the signals it collects from the sensors to act on its environment.

- Artificial intelligence is the study which allows computers to think and act like humans. That is, adapting to new conditions without explicit human programming, abilities to solve novel problems and most importantly, acting rationally, making them human-like.

- Rationality is the act of doing the right thing. When an agent adopts rationality, it strives to pick the best options or the closest to the answers in cases where there is no direct answer.

- Autonomy is the act of changing states, adapting to environments and carrying out new actions without explicit human programming.

- Reflex agents are agents that make decisions based only on the current percepts with no reflections on the past state of the system or the consequences of the actions.

- Model based agents are agents that save the states of the environments and also refer to the initial state of the environment before making decisions based on that. They are Stateful agents.

- Goal based agents are agents that do not only check on the initial state of the environment but consider the consequences of every action they carry out before making a decision.

- Utility based agents are agents that do not only consider the consequences of their actions before acting but they go further to determine how correct these solutions and decisions are before making a decision.

- Learning agents are agents who learn from experience and modify their performance over time. All the above mentioned agents can become learning agents. [1]

b)

Scenario	Performance	Environment	Actuators	Sensors
A robot soccer player	Winning the game, number of goals	A soccer field with defined	legs for movement, kicking mechanism	cameras to spot team mates, court

	scored, minimizing goals conceded, time of ball possession, passing accuracy, running speed.	boundaries, goal post at each end, a ball, teammates, referee and opposing players.	to shoot or pass the ball, a wireless transmitter for communication with teammates, an accelerator that enables the robot to speed up.	boundaries and goal posts, infrared sensors for detecting the proximity of the ball and other bodies on the court, wireless receivers to get information from teammates.
An agent that reviews and approves the courses in a degree plan at CMU-Africa	Accuracy and correctness in verifying all degree requirements, Efficiency and speed of the review process, consistency in applying academic rules to all plans, clarity of feedback for rejected plans.	University academic records, course catalogue, degree requirement document, university academic policies.	An action system that labels degree plans as accepted or rejected, automated email system for communication, a feedback system for giving feedback, a system to update the students record.	An interface that reads and analyses a student's degree plan.

Environment Properties

1) **A robot soccer player:**
Due to the constant movements and constant changes that take place in this environment added to the fact that just just part of this system can be viewed by the robot's sensors at a given time, this system is dynamic and partially observable respectively. This system is non-deterministic given that there are no fixed reactions to signals the sensors pick-up, their next moves are never 100% predictable.
This is a multi-agent (made of many agents), known (Soccer guided by a series of established rules) system with a blend of both discrete(number of goals scored, number of fouls) and continuous features (running distance, speed, ball trajectory).

2) **An agent that reviews and approves the courses in a degree plan at CMU-Africa.**
This is a static (it follows strict underlying policies unless updated), single agent (this environment consists of one agent only), deterministic (everything perceived has a specific known reaction) and completely observable (every part of this environment is perceived at once) environment made up of entirely discrete features.

c) **Agent Architectures**

1) A thermostat controlling the temperature of a room is a simple reflex agent. The thermostat only relies on the perceived temperature of the room to decide if it turns on the heater or the cooler.

2) A robot vacuum cleaner navigating the house with furniture is a model based reflex agent. This agent stores the initial state of the house (Clean state) and any object that is not in the initial state gets sucked up.
3) A self-driving car making decisions at intersections is a utility based agent. It needs to make complex decisions not only relying on the initial state of the road or what path leads to its destination but analyse all the correct options, consider the consequences of any actions it makes and choose the safest actions to take. Ensuring a smooth drive without hitting on other cars, people, and objects on the road and also respecting traffic lights and stop points.

d) **Pseudocode Implementation for the agent program in Scenario 1.**

```
FUNCTION SIMPLE-REFLEX-THERMOSTAT-AGENT(current_temperature):
    CONSTANT DESIRED_TEMPERATURE = 70°F
    CONSTANT COMFORT_RANGE = 2°F

    CONSTANT LOWER_BOUND = DESIRED_TEMPERATURE - COMFORT_RANGE
    CONSTANT UPPER_BOUND = DESIRED_TEMPERATURE + COMFORT_RANGE

    IF current_temperature < LOWER_BOUND THEN
        RETURN "TURN_ON_HEAT"
    ELSE IF current_temperature > UPPER_BOUND THEN
        RETURN "TURN_ON_COOLING"
    ELSE
        RETURN "DO_NOTHING"

END FUNCTION
```
[2]

QUESTION 2

2.1)

- The state space of an environment is the complete set of every possible state, that is, configuration, an environment can be in. For example, in the case of the game ‘SUDOKU’, the state space is the set of all possible positions of numbers.

- The initial state of an agent is the state of the agent when it begins its task.

- Actions are the set of actions available to an agent or that can be executed in a given state.

- The transition model is a description of how actions affect the state. It takes the current state and tells you what the state will be after a particular action. Therefore it states the cause and effects of an agent’s everymove.

- The goal state is the objective of an activity. It might not always be one specific state, just any state that certifies certain properties.

- The action cost is the price of every move you make. For example, leaving Rwanda to Cameroon by air, some of the action costs will be: cost of the flight ticket and time of travel.

- The path cost is the total cost of every sequence of action you make as you work towards achieving the goal. It is obtained by adding up every price (action cost) that makes up every path.

- A tree search is on which doesn't keep a memory of all the states it has already visited. It explores paths without checking if they lead to a state it has seen before through a different route without closed loops.

- Time complexity is a way to monitor how the execution time of an algorithm grows with the size of the problem.

2.2
2.2.1. **Report.**

Objective

The objective of this experiment was to compare the performance of the Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms for solving the 8-puzzle problem with performance metrics; the number of moves and the total execution time.

Methodology:

This solution starts by establishing the foundational logic by defining the final `GOAL_STATE` and providing a `get_neighbors` function to find all valid moves from a given position and specifying the various number of moves to take to go left, right, up or down. The function works by locating the empty tile (0) and applying numerical index offsets to generate potential new board states for each possible move. It validates each generated state by ensuring the move remains within the 3x3 grid and by preventing illegal wrapping moves across the edges of the board by specifying the ILLEGAL moves.

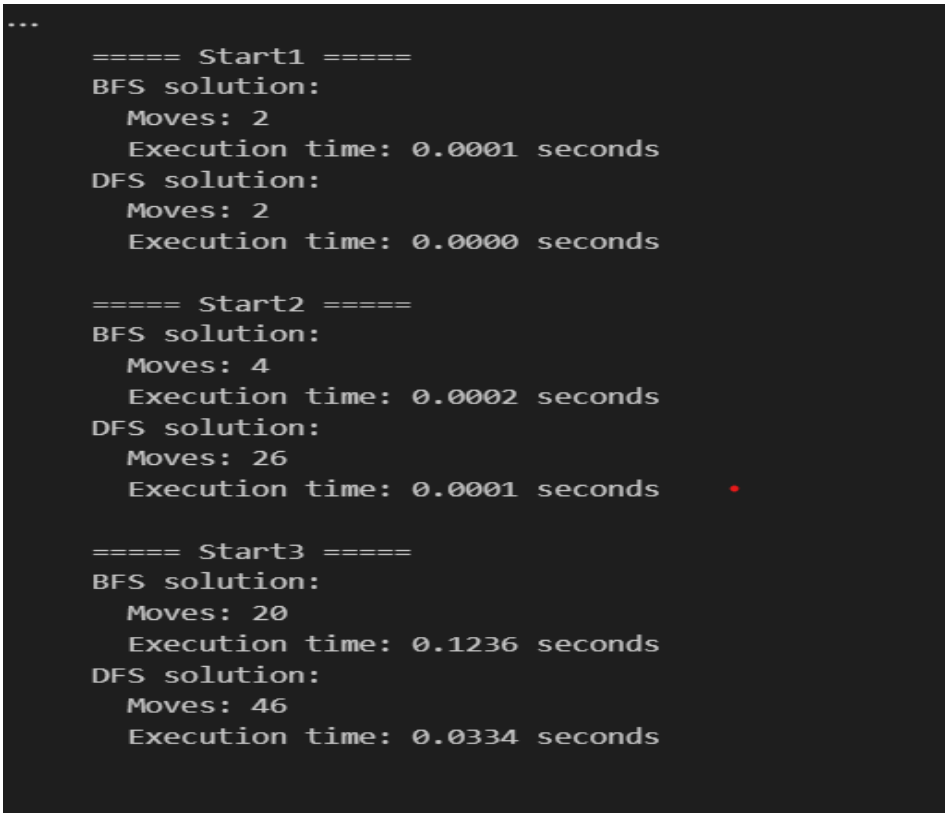
To implement the Breathe First Search algorithm, a function called `bfs` was created, using a queue to explore puzzle states and a `visited` set to prevent redundant processing. The algorithm exhaustively explores the puzzle level by level using the First-In, First-Out (FIFO) order, ensuring all shorter paths are checked before moving to longer ones. This approach guarantees that the first time the goal state is reached, the returned path represents the optimal solution with the minimum number of moves.
As for the Depth First Search algorithm, the `dfs` function uses a stack to manage nodes, causing the search to explore as deeply as possible down one path before backtracking. To prevent infinite loops in cycles and to constrain the search space, the function uses a `visited` set and a `depth_limit` parameter. This memory-efficient, Last-In, First-Out (LIFO) approach prioritizes finding a solution quickly over guaranteeing that the solution is optimal in length.

These two algorithms were implemented on three predefined 8-puzzle start states of varying difficulty: 'Start1' (easy), 'Start2' (medium), and 'Start3' (hard). Performance was evaluated based on execution time, measured by recording the wall-clock duration of each function call using Python's `time` module, and the number of moves, calculated as the solution path length minus one. To ensure a fair comparison and guarantee a solution for the most difficult test case, the DFS algorithm was implemented with a depth limit of 9, a value corresponding to the optimal number of moves required for the 'Start3' puzzle.

Results:

		BFS	DFS
Start1	Moves	2	2
	Execution time/s	0.0001	0.0000
Start2	Moves	4	26
	Execution time/s	0.0002	0.0001
Start3	Moves	20	46
	Execution time/s	0.1236	0.0334

Table of the results showing the execution time and number of moves per algorithm in every start case.



Picture of the results showing the execution time and number of moves per algorithm in every start case.

2.2.2
According to the results of the code, the depth limits after running the DFS algorithm are:

Start1: Depth Limit is 2
Start2: Depth Limit is 26
Start3: Depth Limit is 46

These results could be regarded as the optimal depths but from the knowledge of the depth first search algorithm acting like a maze runner who picks up a path and follows it to the end without thoughts of better options. I could argue that DFS does not yield any results what can be confidently concluded upon as the optimal path; it only yields a possible path until I came across the concept of depth limited search algorithm which yields the same results as the Breath first search algorithm when the depth limit is set to the same limits yielded by the breath first search algorithm. Using this method, the optimum depths limits become:
Start1: Depth Limit is 2
Start2: Depth Limit is 4
Start3: Depth Limit is 20.[3]

2.3 Analysis

2.3.1.
The breath first search algorithm produced solutions with the fewest moves according to the results of our code. This is because it explores the puzzle level by level, hence guaranteeing that it finds the shortest possible path.

2.3.2.
On analyzing how the algorithms scaled with problem difficulty, Breadth-First Search (BFS) performed worse, looking at the execution time from 0.0001s in 'Start1' to 0.1236 in 'Start3'.While the memory requirement for Depth-First Search (DFS) grew only linearly with the solution depth, making it far more scalable for deeper problems.

2.3.3.

These experimental results align with the theoretical properties of BFS and DFS.

BFS guarantees the finding of optimal solution, this is as a result of its level-by-level traversal. However, its major drawback is its exponential O(b^d) space complexity was also evident. As the puzzle difficulty increased, the number of nodes BFS had to store in its queue grew exponentially, causing a significant rise in execution time and demonstrating why it scales poorly.

DFS did not directly yield an optimal depth limit but looking at situations of scaling, it demonstrated superior efficiency due to its linear O(d) space complexity. This performance shows why it is optimal for problems where memory is a concern. The use of a depth limited search algorithm constrains DFS to find the optimal path, combining its efficiency with a guarantee of optimality. [4]

BIBLIOGRAPHY

[1] "Artificial Intelligence: A Modern Approach, 4th US ed." Accessed: Sept. 01, 2025. [Online]. Available: https://aima.cs.berkeley.edu/

[2] "Simple Reflex Agents in AI," GeeksforGeeks. Accessed: Sept. 22, 2025. [Online]. Available: https://www.geeksforgeeks.org/artificial-intelligence/simple-reflex-agents-in-ai/

[3] Mahesh Huddar, *Solved Example Depth Limited Depth First Search (DLDFS) in Artificial Intelligence Mahesh Huddar*, (Dec. 16, 2022). Accessed: Sept. 21, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=P7WQUBLKDmo

[4] A. K. Mishra, "Understanding Search Algorithms: BFS, DFS, Depth-Limited Search, and IDDFS," Medium. Accessed: Sept. 21, 2025. [Online]. Available: https://medium.com/@ankitkm1015/understanding-search-algorithms-bfs-dfs-depth-limited-search-and-iddfs-2ff0eb1387f3