

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240030037 |
| **NAME** | Hariharan.N |
| **PROJECT TITLE** | Text based adventure game using bfs search |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

* **INTRODUCTION**
  + **Artificial Intelligence (AI)** enables machines to perform tasks that require human-like reasoning, learning, and decision-making.  
    In gaming, AI is widely used for **pathfinding and intelligent navigation**.
  + This project implements a **Text-Based Adventure Game** using the **Breadth-First Search (BFS)** algorithm.  
    The aim is to simulate an AI agent that explores rooms (as graph nodes) and finds the **shortest path** to the **Treasure Room**.
* **PROBLEM STATEMENT**
  + This project implements a **Text-Based Adventure Game** using the **Breadth-First Search (BFS)** algorithm.  
    The aim is to simulate an AI agent that explores rooms (as graph nodes) and finds the **shortest path** to the **Treasure Room**.
  + **Expected Result:**
    - BFS identifies the shortest and most efficient path.
    - The player can explore manually or use AI auto-navigation.
  + **Future Possibilities:**
    - Add obstacles, locked rooms, and puzzles.
    - Extend with A\* Search or a graphical interface (Tkinter/Pygame).
* **THEORETICAL BACKGROUND**
  + **DFS:** Explores deep paths, may not find shortest path.
  + **BFS:** Explores level by level, **guarantees shortest path**. ✅
  + **Dijkstra / A\*:** Weighted graphs; overkill for unweighted room navigation.
  + **Justification for Choosing BFS**
    - The game’s map can be represented as an **unweighted graph**, so BFS ensures the shortest path.
    - BFS is **guaranteed to find a solution** if one exists.
    - Other algorithms like DFS may fail to find the shortest path or may explore unnecessary paths.
    - Weighted algorithms (Dijkstra, A\*) are **unnecessary** here, as all moves are considered equal.
* **ALGORITHM EXPLANATION WITH EXAMPLE**
  + **Breadth-First Search (BFS)** is a **graph traversal algorithm** that explores nodes level by level:
    - **Start** at the source node (starting room).
    - **Add** the source node to a queue and mark it as visited.
    - **While the queue is not empty**:  
      a. Remove the first node from the queue.  
      b. If it is the goal node, stop and construct the path.  
      c. Otherwise, add all unvisited neighbors to the queue and mark them visited.
    - **Repeat** until the goal is found or all nodes are explored.
  + **BFS Example**

**Entrance -- Hall -- Kitchen**

**| |**

**Library Garden**

**|**

**Tower -- Treasure** **Room**

* + **Execution Steps:**
    - **Queue: [Entrance] → Explore Hall & Library**
    - **Queue: [Hall, Library] → Explore Kitchen & Garden**
    - **Queue: [Garden, Tower] → Explore Treasure Room**
    - **Shortest Path:**
    - **Entrance → Hall → Garden → Tower → Treasure Room**

**IMPLEMENTATION AND CODE**

from collections import deque

class Room:

def \_\_init\_\_(self, name, desc):

self.name = name

self.desc = desc

self.exits = {}

def connect(self, direction, other\_room):

self.exits[direction] = other\_room

def describe(self):

lines = [f"\n📍 You are in the {self.name}.", self.desc]

if self.exits:

lines.append("🚪 Exits: " + ", ".join(f"{d} -> {r.name}" for d, r in self.exits.items()))

else:

lines.append("No exits from here.")

return "\n".join(lines)

class Game:

def \_\_init\_\_(self):

self.rooms = {}

self.create\_world()

self.player\_room = self.rooms['Entrance']

def create\_world(self):

# Create rooms

r = lambda n, d: Room(n, d)

entrance = r("Entrance", "An old wooden door creaks behind you. The hall lies ahead.")

hall = r("Hall", "Portraits of knights watch silently.")

kitchen = r("Kitchen", "You smell something burnt. A rusty knife lies on the counter.")

library = r("Library", "Dusty books line the shelves. Something glimmers between them.")

garden = r("Garden", "Overgrown vines cover the walls.")

tower = r("Tower", "A spiral staircase winds upward.")

treasure = r("Treasure Room", "💎 You found the treasure chest filled with gold!")

# Connect rooms (graph edges)

entrance.connect('north', hall); hall.connect('south', entrance)

hall.connect('east', kitchen); kitchen.connect('west', hall)

hall.connect('west', library); library.connect('east', hall)

hall.connect('north', garden); garden.connect('south', hall)

garden.connect('up', tower); tower.connect('down', garden)

tower.connect('north', treasure); treasure.connect('south', tower)

for room in [entrance, hall, kitchen, library, garden, tower, treasure]:

self.rooms[room.name] = room

def bfs\_shortest\_path(self, start\_name, goal\_name):

"""Find shortest path using BFS"""

start, goal = self.rooms[start\_name], self.rooms[goal\_name]

queue = deque([start])

parent = {start.name: None}

visited = {start.name}

while queue:

current = queue.popleft()

if current.name == goal.name:

path = []

while current:

path.append(current.name)

current = self.rooms[parent[current.name]] if parent[current.name] else None

return path[::-1]

for direction, neighbor in current.exits.items():

if neighbor.name not in visited:

visited.add(neighbor.name)

parent[neighbor.name] = current.name

queue.append(neighbor)

return None

def play(self):

print("🎮 Welcome to the Adventure Game!")

print("Type directions (north, south, east, west, up, down) to move.")

print("Type 'auto' to find the shortest path to the treasure.")

print("Type 'exit' to quit.\n")

while True:

print(self.player\_room.describe())

command = input("\n➡️ What do you want to do? ").strip().lower()

if command == "exit":

print("👋 Thanks for playing! Goodbye!")

break

elif command == "auto":

path = self.bfs\_shortest\_path(self.player\_room.name, "Treasure Room")

print("\n🧭 BFS Path to Treasure:")

print(" -> ".join(path))

if self.player\_room.name != "Treasure Room":

print("\nFollowing the path automatically...")

for step in path[1:]:

print(f"Moving to {step}...")

print("\n💎 Congratulations! You reached the Treasure Room!")

break

elif command in self.player\_room.exits:

self.player\_room = self.player\_room.exits[command]

if self.player\_room.name == "Treasure Room":

print("\n💎 You found the Treasure Room! Game Over!")

break

else:

print("❌ Invalid command! Try a valid direction or 'auto'.")

# Run the game

if \_\_name\_\_ == "\_\_main\_\_":

game = Game()

game.play()

* **OUTPUT**
  + **Game Start Screen:**
    - Shows the welcome message, instructions for manual and auto mode.
    - Example:A screen shot of a computer

      AI-generated content may be incorrect.
  + **Manual Navigation Example:**
    - Player chooses a directionA computer screen shot of a black screen

      AI-generated content may be incorrect.
  + **Auto Navigation Example (BFS):**
    - Player types auto to find shortest path.
    - BFS finds and follows
    - path step-by-step:A screenshot of a computer program

      AI-generated content may be incorrect.
  + **Explanation:**
    - The BFS algorithm explores all rooms level by level.
    - Ensures the **shortest path** from Entrance to Treasure Room.
    - Manual mode allows **player exploration**, auto mode demonstrates **algorithm efficiency**.
    - Current room description and available exits are displayed
* **RESULTS AND FUTURE ENHANCEMENT**
  + **Result**
    - BFS successfully finds the **shortest path** in all tested room graphs.
    - Automatic navigation simulates **real-time movement** using step delays.
    - Manual mode allows **user-driven exploration**.
    - Algorithm handles multiple branches and avoids revisiting rooms.
  + **Future Enhancements**
    - **Obstacles and Collectibles:**
      * **Introduce keys, locked doors, traps, or monsters to make gameplay challenging.**
      * **BFS can be modified to avoid obstacles or collect items before reaching the goal.**
    - **Weighted Paths:**
      * **Assign weights to rooms or paths (danger level, cost, time).**
      * **Use Dijkstra or A\* to find optimal paths in weighted scenarios.**
    - **Randomized Map Generation:**
      * **Generate new maps each time for replayability.**
      * **BFS can still adapt to find paths in dynamically generated graphs.**
    - **Graphical Interface:**
      * **Upgrade from text-based to a GUI using Python’s Tkinter or Pygame.**
      * **Animate player movement along the BFS path visually.**
    - **Multiple Goals or Quests:**
      * **Include multiple treasure rooms or side quests.**
      * **BFS can compute paths for multiple goals efficiently.**

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | **https://github.com/Hariharan-0607/Ai-mini-project.git** |

**REFERENCES**

* GeeksforGeeks. *Breadth First Search (BFS) Algorithm in Python.*  
  https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
* Python Software Foundation. *Python 3 Official Documentation.*  
  https://docs.python.org/3/
* Real Python. *Build a Text Adventure Game with Python.*  
  https://realpython.com/python-adventure-game/
* TutorialsPoint. *Python Classes and Objects.*  
  https://www.tutorialspoint.com/python/python\_classes\_objects.htm
* Medium (Towards Data Science). *Understanding Breadth-First Search (BFS) in Graph Theory.*  
  https://towardsdatascience.com/