VITYATHI PROJECT REPORT

Name : Harshita thakur
Registration number : 25MIM10024
Course: python essentials
Academic year :-2025-26

Project title

Text-Based Adventure Game

Introduction:-

a classic console experience where imagination drives the journey. In this project, players explore mysterious rooms, uncover secrets, and solve challenges by typing simple commands like go north, open door, or take key.
This game demonstrates how to build an interactive story using python fundamental

Problem Statement

Design a text-based adventure game where the player navigates through different rooms by typing commands. The game should use dictionaries to define locations and their connections, handle user input to interpret actions, and apply conditional logic to manage movement, locked paths, and game progression
.

Functional Requirements:-

These are the main things the game should be able to do.
Major Functional Parts

1. **Moving Between Rooms**

The player can type commands like go north or go east to move.

The game changes the current room based on the command.

2. **Input and Output**
    - The game takes text input from the player.
    - It shows room descriptions, messages, and results as output.
3. **Game Rules**

- The game checks if doors are locked or open.
- It decides if the player wins, loses, or makes a wrong move.

Non-Functional Requirements:-

These are the qualities that make the game work well, even if they are not direct features.

1. **Performance**

    The game should run quickly and respond to user commands without delay.
2. **Usability**

    The game should be easy to play with simple text commands that are clear to understand.
3. **Reliability**

    The game should not crash and should always give the correct output for valid commands.

4. **Maintainability**

- The code should be simple and organized so that new rooms or features can be added easily.

Error Handling Strategy

- If the player types a wrong command, the game should show a friendly error message instead of stopping.

Logging or monitoring:-

The game should keep track of important actions (like moves or errors) so the developer can check how the game is working.

System architecture

1. **Input Module**

Takes commands from the player (like go north, open door, quit).

2. **Game Logic Module**

Decides what happens based on the command.

Checks if the move is valid, if a door is locked, or if the player has won.

3. **Output Module**

- Shows the room description, error messages, or winning message back to the player.

Workflow (How They Connect)

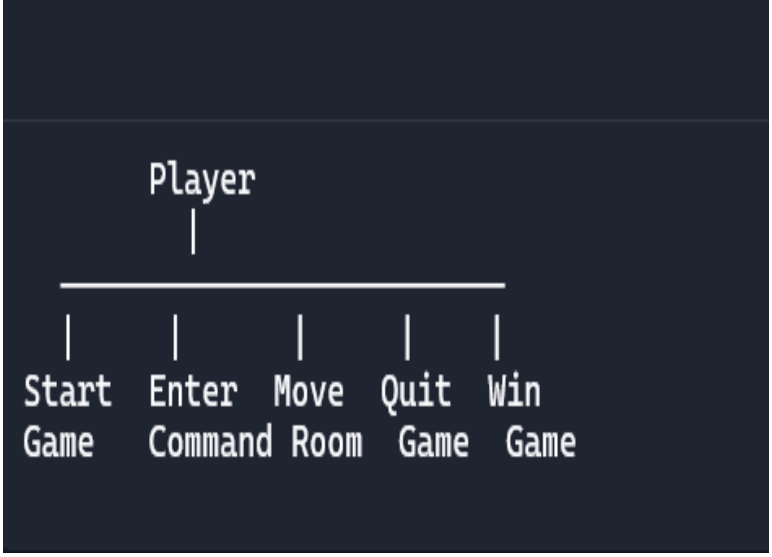
- Player types a command → input module
-
- Command goes to → Game Logic Module
-
- (checks rules and updates state)
-
- Result goes to → Output Module (prints
- message to the player)
-
- Loop continues until the player quits or wins.
-

layer Input → Input Module → Game Logic Module → Output Module → Player sees result

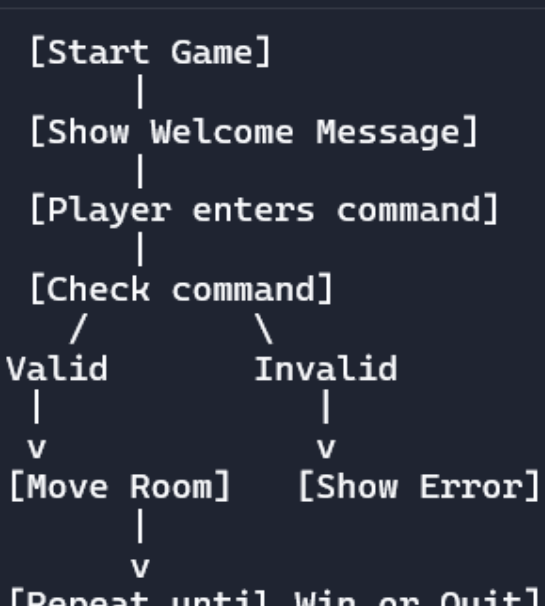
layer Input → Input Module → Game Logic Module → Output Module → Player sees result

7. Design Diagrams

USE CASE DIAGRAMS

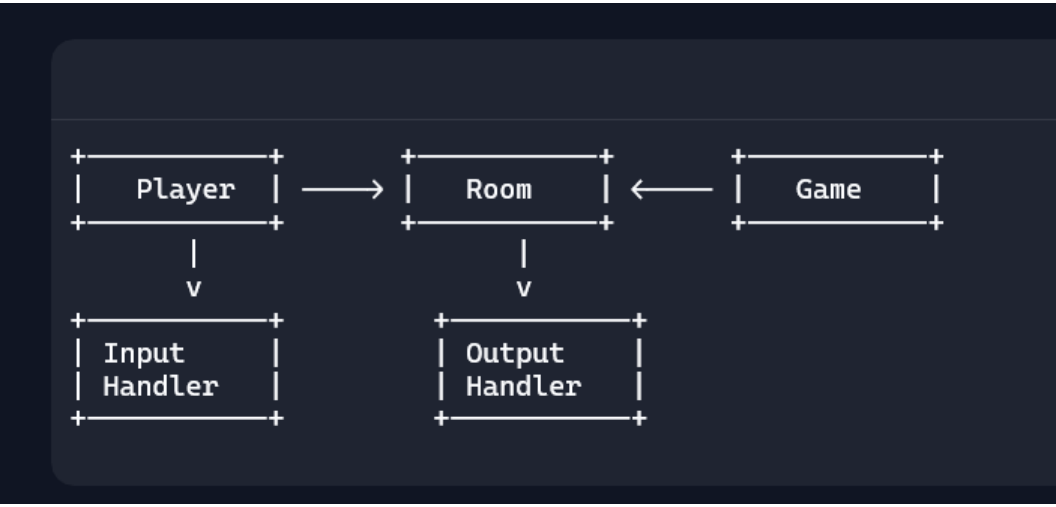


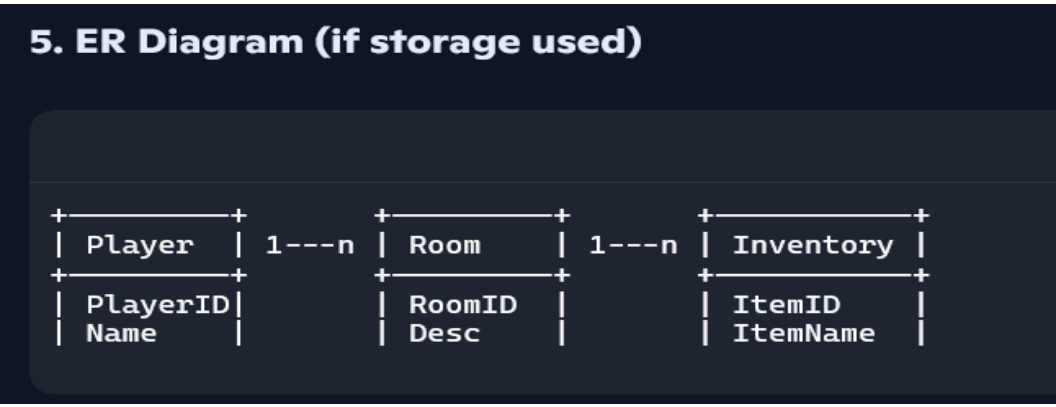
2. Workflow Diagram (Flowchart)



. Sequence Diagram
Player -> Input Module: "go north" Input Module -> Game Logic: check command Game Logic -> Output Module: result Output Module -> Player: "You are in the kitchen"

CLASS COMPONENT DIAGRAM:





DESIGN DESIGN AND RATIONALE
Decision: Pick a simple theme (e.g., haunted house, treasure hunt, space mission).

Rationale: Easy to imagine and write about, keeps the game fun**.**

Rationale: Easy to imagine and write about, keeps the game fun.

1. **Player Choices**
   Decision: Use simple commands like *“go north”*, *“pick up key”*, or menu options.
   Rationale: Makes the game easy to play without confusing the player.
2. **World Layout**
   Decision: Create rooms or places connected by directions (north, south, east, west).
   Rationale: Simple map design helps players explore step by step.
3. **Story Progression**
   Decision: Add different paths and endings depending on choices.
   Rationale: Gives players control and makes them want to play again.
4. **Characters**
   Decision: Include a few NPCs (non-player characters) like a guard, shopkeeper, or monster.
   Rationale: Adds interaction and makes the world feel alive.
5. **Challenges**

Decision: Add puzzles (find a key, solve a riddle) or simple battles.

Rationale: Keeps the game interesting and tests problem-solving.

6. **Descriptions**

   Decision: Write short but clear text to describe rooms, items, and actions.

   Rationale: Since there are no graphics, words must paint the picture.

7. **Help System**

   Decision: Add a “help” command to show possible actions.

   Rationale: Supports beginners and reduces frustration.

IMPLEMENTATION DETAILS:- Game Loop

- The game runs inside a continuous loop.
- Each cycle of the loop:
  1. Shows the current room description.
  2. Asks the player for a command.
  3. Processes the command.
  4. Updates the game state (location, inventory).
  5. Checks win/lose conditions.
- The loop ends when the player wins, loses, or types quit.

3. World / Rooms

- The game world is divided into rooms or locations.
- Each room has:

  A name (e.g., Hall, Kitchen, Dungeon).
  Connections to other rooms (north, south, east, west).

  Items or characters inside (e.g., key, treasure, monster).

  4. Player State
- CURRENT ROOM**:** Tracks where the player is.
- Inventory: A list of items the player has collected.
- These two elements define the player's progress in the gam

. Commands

- Players interact using simple text commands:
  - Movement: go north go south, etc.
  - Item interaction:`take key drop item`.
  - Help: `help` shows available commands.
  - Quit: `quit` ends the game.

. Input Processing

- The game checks the player's input against possible commands.
- If the input matches a valid command → perform the action.

- If the input is invalid → show an error message or hint.

7. Win / Lose Conditions

- **win:** Player achieves the main goal (e.g., finds treasure, escapes the dungeon).
- **Lose:** Player encounters a losing event (e.g., meets a monster, runs out of moves).
- These conditions are checked after each action.

8. Game Flow (Step by Step)

1. Display the description of the current room.
2. Ask the player for a command.
3. Process the command (movement, item collection, etc.).
   - Update the player's state (location, inventory).
4. Check if the player has won or lost.
5. Repeat until the game ends.
6.

Screenshot \result

```
rooms = {
    "Hall": {"north": "Kitchen", "east": "Dining Room", "item": "key"},
    "Kitchen": {"south": "Hall", "item": "monster"},
    "Dining Room": {"west": "Hall", "north": "Garden", "item": "treasure"},
    "Garden": {"south": "Dining Room"}
}

# Player state
current_room = "Hall"
inventory = []

# Show instructions
def show_instructions():
    print("Text Adventure Game")
    print("Commands: go [direction], take [item], help, quit")

# Show current status
def show_status():
    print("\nYou are in the", current_room)
    if "item" in rooms[current_room]:
        print("You see a", rooms[current_room]["item"])
    print("Inventory:", inventory)

# Main game loop
show_instructions()

while True:
    show_status()
    command = input("\nEnter your move: ").lower()

    # Quit command
```

```
# Quit command
if command == "quit":
    print("Thanks for playing!")
    break

# Help command
elif command == "help":
    show_instructions()

# Movement command
elif command.startswith("go "):
    direction = command.split()[1]
    if direction in rooms[current_room]:
        current_room = rooms[current_room][direction]
    else:
        print("You can't go that way!")

# Take item command
elif command.startswith("take "):
    item = command.split()[1]
    if "item" in rooms[current_room] and item == rooms[current_room]["item"]:
        inventory.append(item)
        print("You picked up the", item)
        del rooms[current_room]["item"]
    else:
        print("There is no", item, "here!")

# Win condition
if "treasure" in inventory:
    print("\nCongratulations! You found the treasure. You win!")
    break
```

```
Text Adventure Game
Commands: go [direction], take [item], help, quit

You are in the Hall
You see a key
Inventory: []

Enter your move:  take key
You picked up the key

You are in the Hall
Inventory: ['key']

Enter your move:  go east

You are in the Dining Room
You see a treasure
Inventory: ['key']

Enter your move:  take treasure
You picked up the treasure

Congratulations! You found the treasure. You win!
```

11. Testing Approach

To check whether the game was working properly, I played it several times using different commands. I tried moving in all directions, picking up items, and even entering wrong inputs to see how the program

reacted. I also tested both the winning path (collecting the treasure) and the losing path (meeting the monster). This helped me confirm that the loop, inventory, and win/lose conditions were functioning as expected.

12. Challenges Faced

One of the main difficulties was making sure the game understood the commands correctly. At first, the program didn't respond well to unexpected inputs or spelling mistakes. Another challenge was setting up the room connections in a way that made sense, because if the dictionary wasn't written properly, the game would crash. Handling the inventory also took some effort, especially removing items from rooms once they were picked up.

13. Learnings & Key Takeaways

Through this project, I learned how loops and conditional statements can control the flow of a program. I also understood how dictionaries and lists can be used to store data like rooms and items. Another important takeaway was the need for clear instructions and error handling, because without them the game becomes confusing for the player. Overall, the project gave me confidence in combining basic programming concepts to create something interactive.

14. Future Enhancements

If I continue working on this game, I would like to add more rooms and items to make the adventure longer. Another idea is to include puzzles or locked doors that require keys to open. I could also design multiple endings so that the player's choices lead to different outcomes. Improving the command system to accept more flexible inputs (like "pick up key" instead of only "take key") would make the game more user-friendly. Finally, adding a save/load feature would allow players to continue their progress later.

References:

1. W3resource, *Python Project – Text-Based Adventure Game*.
2. GeeksforGeeks, *Building a Text-Based Adventure Game with SpaCy: A Step-by-Step Tutorial*.
3. Studytonight, *Text-Based Adventure Game Python Project*.
4. GitHub – nikhilcigmallu, *Adventure Game Text-based Python Project*.
5. GitHub Topics, *Text-Based Adventure Game*.