|  |
| --- |
| ***A signed copy of this form must be submitted with every assignment.***  ***If the statement is missing your work may not be marked.***  ***For Level 7 assignments, please use the separate Level 7 Candidate***  ***Statement of Own Work form instead of this one.*** |

Student Declaration

I confirm the following details:

|  |  |
| --- | --- |
| **Candidate Name:** | Shehani Kaveenya Mukadange |
| **Candidate ID Number:** | 212262 |
| **Qualification:** | Qualification: Level 3 International Foundation Diploma for Higher Education Studies |
| **Unit:** | Introduction to Python Programming |
| **Centre:** | UCL Campus Sri Lanka |
| **Word Count:** | 2002 |
| I have read and understood both NCC Education’s *Academic Misconduct Policy* and the *Referencing and Bibliographies* document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly.  I confirm that I have not exceeded the stipulated word limit by more than 10%.  I confirm that this is my own work and that I have not colluded or plagiarized any part of it. | |
| **Candidate Signature:** | Shehani Kaveenya |
| **Date:** | 30th April 2025 |

Task 01 – Decomposition and Design

**Specification, Outcomes & the Success Criteria**

1. The game must ask the player to input the name of their characters.
   * The game must prompt the player to input their character’s name at the beginning.
2. The player must be able to move between at least 10 different positions.
   * The player must be able to move between a minimum of 10 different levels.
3. The player must have a bag to store up to 4 collected items.
   * The player must have a bag that stores up to 4 items collected during the journey.
4. Each position must display a description and options for movement (Forward & Back)
   * Each position must display description and options.
5. The player must interact with levels, complete tasks to obtain items.
   * The player must interact with levels to solve tasks and collect items.
6. Some paths must be locked and require specific items to unlock.
   * Some paths must be locked and require a specific item to proceed.
7. The game must end successfully when the player reaches the final level and display a winning message.
   * The game must display a winning message when the player reaches the treasure room.
8. The game must be designed using Object Oriented Programming, particularity using at lease on class.
   * The program must use OOP (player class and methods).

**Game Story and Layout**

* The player embarks on a journey through mysterious forests, riverbanks, villages and castles to find a legendary treasure hidden deep within a castle’s treasure room. Along the way, the player must solve riddles, collect items and overcome locked obstacles.

**Level Structure**

1. Level 1: Forest Entrance
2. Level 2: Clearing
3. Level 3: Riverbank
4. Level 4: Bridge
5. Level 5: Village ­Entrance
6. Level 6: Village Market
7. Level 7: Village Inn
8. Level 8: Castle Gate
9. Level 9: Castle Hallway
10. Level 10: Treasure Room

**Important Items**

1. Map – Helps navigate
2. Lantern – Needed for dark places
3. Rope – For crossing hazards
4. Potion - Boosts strength
5. Sword – Defends against threats
6. Shield, Key, Crown

**Data Structures Required**

* Class (Player) – To manage player information (name, position, bag)
* List (Bag) – To store up to 4 collected items.
* Dictionary (World) – To store details about each level (Description, available, moves, task, item)

**Decompose the game into sub problems**

* Get user input – Handle user commands (Move, interact, check bag)
* Movement handling – Move players between positions based on available options.
* Interaction handling – Allow player to interact with levels and solve tasks.
* Inventory management – Add or Remove items from the bag based on interactions.
* Locked door handling – Check if the player has the required item to unlock a path.
* Winning the Game – Detect when the player reaches the final level and display a winning messages.

**Structure Diagram**

7

**Pseudocode**

START

DEFINE Class Player:

    INIT with name, initial position at "Level 1: Forest Entrance", bag is empty

    METHOD add\_to\_bag(item):

        IF bag contains less than 10 items:

            Add item to bag

            PRINT item acquired

ELSE:

            PRINT bag full message

    METHOD check\_bag():

        IF bag not empty:

            PRINT contents of bag

        ELSE:

PRINT bag is empty

FUNCTION get\_level\_number(position):

    SPLIT position string and RETURN level number

FUNCTION display\_world(world, player):

    GET current location from world using player's position

    EXTRACT level number

    PRINT level number and description

    IF item exists at location:

PRINT item information

PRINT possible moves

FUNCTION move(world, player, direction):

    IF valid direction selected:

        CHECK if next space is locked

            IF locked:

IF player has Key:

                UNLOCK location

                REMOVE Key from player's bag

MOVE player to next room

                ELSE:

                    PRINT door locked message

RETURN

            ELSE:

                MOVE player to next room

PRINT new position

ELSE:

    PRINT invalid move message

FUNCTION interact(world, player):

    GET current location

    IF location has item and task:

        PRINT task

GET player answer

        IF answer correct:

            ADD item to bag

            REMOVE item, task, and answer from location

        ELSE:

PRINT incorrect answer message

ELSE:

    PRINT nothing to interact message

DEFINE world as a DICTIONARY:

    - Location names are the keys

    - Descriptions, options, tasks, answers, items, and locked states are the values

FUNCTION main():

    GET player name

CREATE Player object

    PRINT welcome message

    WHILE player is not at "Level 10: Treasure Room":

        CALL display\_world(world, player)

        ASK player for action (move/interact/check bag)

        IF action is move:

ASK for direction (forward/back)

            CALL move(world, player, direction)

        ELIF action is interact:

            CALL interact(world, player)

        ELIF action is check bag:

CALL player.check\_bag()

        ELSE:

            PRINT invalid action message

    PRINT win message

    PRINT end of game message

IF script is run directly:

    CALL main()

END

Task 02 – Implementation

**Python Code**

import random

class Player:

    def \_\_init\_\_(self, name):

        self.name = name

        self.position = "Level 1: Forest Entrance"

        self.bag = []

    def add\_to\_bag(self, item):

        if len(self.bag) < 10:

            self.bag.append(item)

            print(f"You obtained {item}.")

        else:

            print("Your bag is full!")

    def check\_bag(self):

        if self.bag:

            print("Your bag contains:", ", ".join(self.bag))

        else:

            print("Your bag is empty.")

def get\_level\_number(position):

    """Extract and return the level number from the position string."""

    return position.split(":")[0]

def display\_world(world, player):

    location = world[player.position]

    level\_number = get\_level\_number(player.position)

    print(f"\n This is {level\_number}")

    print(location['description'])

    if "item" in location:

        print(f"You see an item here: {location['item']} (Complete the task to get it!)")

    print("Options:")

    for option in location["options"]:

        print(f"- {option}")

def move(world, player, direction):

    if direction in world[player.position]["options"]:

        next\_room = world[player.position]["options"][direction]

        if "locked" in world[next\_room] and world[next\_room]["locked"]:

            if "Key" in player.bag:

                print("You used the Key to unlock the door.")

                world[next\_room]["locked"] = False

                player.bag.remove("Key")

            else:

                print("The door is locked. You need a Key.")

                return

        player.position = next\_room

        print(f"You moved to {player.position}.")

    else:

        print("You can't go that way!")

def interact(world, player):

    location = world[player.position]

    if "item" in location and "task" in location:

        print(location["task"])

        answer = input("Your answer: ").lower()

        if answer == location["answer"].lower():

            player.add\_to\_bag(location["item"])

            del location["item"]

            del location["task"]

            del location["answer"]

        else:

            print("That’s not correct. Try again later.")

    else:

        print("Nothing to interact with here.")

# Define the game world

world = {

    "Level 1: Forest Entrance": {

        "description": "You are at the entrance of a dark forest.",

        "options": {"forward": "Level 2: Clearing"},

        "task": "Task: Solve this riddle - What has roots but never grows?",

        "answer": "Mountain",

        "item": "Map"

    },

    "Level 2: Clearing": {

        "description": "A bright clearing with wildflowers.",

        "options": {"forward": "Level 3: Riverbank", "back": "Level 1: Forest Entrance"},

        "task": "Task: What shines at night but disappears by dawn?",

        "answer": "Star",

        "item": "Lantern"

    },

    "Level 3: Riverbank": {

        "description": "A fast-flowing river blocks your path.",

        "options": {"forward": "Level 4: Bridge", "back": "Level 2: Clearing"},

        "task": "Task: How many legs does a spider have?",

        "answer": "8",

        "item": "Rope"

    },

    "Level 4: Bridge": {

        "description": "The bridge creaks under your weight.",

        "options": {"forward": "Level 5: Village Entrance", "back": "Level 3: Riverbank"},

        "task": "Task: What can run but never walks?",

        "answer": "Water",

        "item": "Potion"

    },

    "Level 5: Village Entrance": {

        "description": "The village gate stands tall.",

        "options": {"forward": "Level 6: Village Market", "back": "Level 4: Bridge"},

        "task": "Task: Unscramble this word: 'drows'",

        "answer": "sword",

        "item": "Sword"

    },

    "Level 6: Village Market": {

        "description": "Traders and shoppers bustle around.",

        "options": {"forward": "Level 7: Village Inn", "back": "Level 5: Village Entrance"},

        "task": "Task: What has hands but can’t clap?",

        "answer": "Clock",

        "item": "Shield"

    },

    "Level 7: Village Inn": {

        "description": "A warm place with crackling fireplaces.",

        "options": {"forward": "Level 8: Castle Gate", "back": "Level 6: Village Market"},

        "task": "Task: Guess the number I'm thinking of (1-5):",

        "answer": str(random.randint(1, 5)),

        "item": "Key"

    },

    "Level 8: Castle Gate": {

        "description": "The grand castle gate is locked.",

        "options": {"forward": "Level 9: Castle Hallway", "back": "Level 7: Village Inn"},

        "locked": True

    },

    "Level 9: Castle Hallway": {

        "description": "Echoes fill the hallway. One last task remains.",

        "options": {"forward": "Level 10: Treasure Room", "back": "Level 8: Castle Gate"},

        "task": "Task: What has a neck but no head?",

        "answer": "Bottle",

        "item": "Crown"

    },

    "Level 10: Treasure Room": {

        "description": "Congratulations! You've found the treasure!",

        "options": {}

    }

}

def main():

    name = input("Enter your character's name: ")

    player = Player(name)

    print(f"Welcome, {player.name}! Your 10-level adventure begins now.")

    while player.position != "Level 10: Treasure Room":

        display\_world(world, player)

        action = input("What do you want to do? (move/interact/check bag): ").lower()

        if action == "move":

            direction = input("Which direction? (forward/back): ").lower()

            move(world, player, direction)

        elif action == "interact":

            interact(world, player)

        elif action == "check bag":

            player.check\_bag()

        else:

            print("Invalid action. Try again.")

    print("You win! The treasure is yours. Glory and riches await!")

    print("Thanks for playing Treasure Quest!")

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

    main()

Task 03 – Testing

**Testing Strategy**

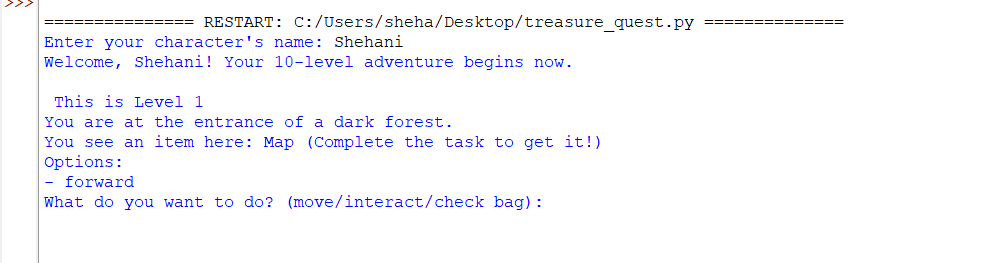
* Normal Testing: Checking correct user moves and item collection.
* Boundary Testing: Trying to add more than 4 items to the bag.
* Invalid input testing: Handling wrong commands.
* Unit Testing: Test each function independently.
* Integration Testing: Checking how movement, interaction, and bag management work together.
* Full system testing: Full play through from beginning to victory.

**Test Log**

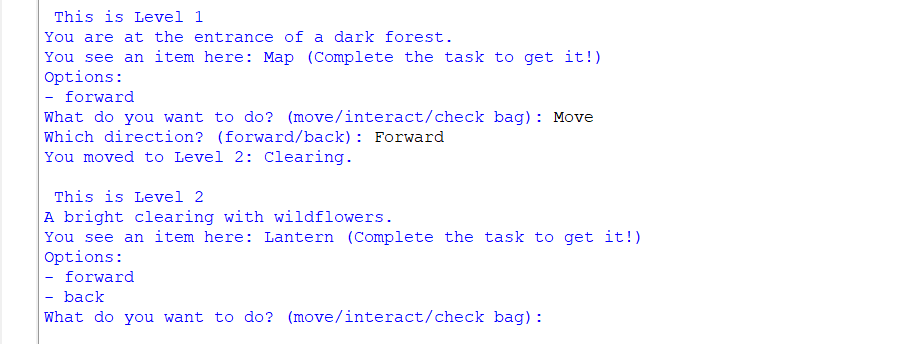
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Purpose | Input | Expected Output | Actual Output |
| 1 | Start Game | Enter character name | Name stored, Game begins | Name accepted, intro shown |
| 2 | Move forward successfully | Move 🡪 Forward | Player moves to Level 1 | Player moved forward |
| 3 | Move backward successfully | Move 🡪 Back | Player returns to Level 2 | Player moved back |
| 4 | Invalid move | Move 🡪 left (no left path) | Error message displayed | Correct error shown |
| 5 | Interact with task | Interact 🡪 Correct answer | Item added to bag | Item added to bag |
| 6 | Interact wrong answer | Interact 🡪 Wrong answer | No item added, retry prompt | No item added |
| 7 | Check bag (empty) | Check bag | Bag empty message | Correct message shown |
| 8 | Check bag (items inside) | Check bag after pickup | List of collected items shown | Items listed |
| 9 | Pick up item 1 | Interact 🡪 Solve task | Item 1 added to bag | Item 1 added |
| 10 | Pick up item 2 | Interact 🡪 solve task | Item 2 added to bag | Item 2 added |
| 11 | Pick me item 3 | Interact 🡪 Solve task | Item 3 added to bag | Item 3 added |
| 12 | Unlock locked Castle Gate | Use key item | Castle gate unlocked | Gate unlocked |
| 13 | Win the game | Reach treasure room | Victory message displayed | Congratulations message |

**Evidence**

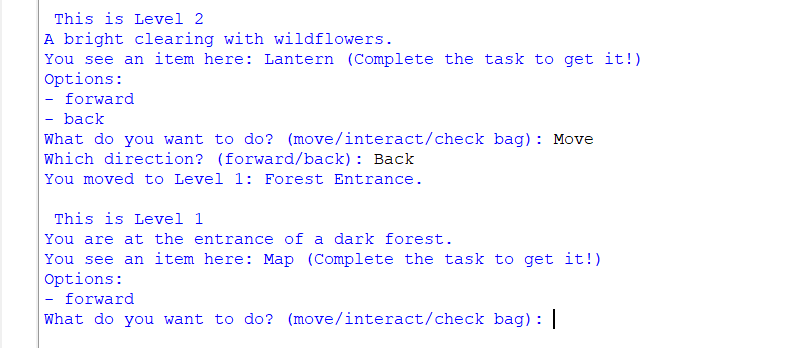
1. **Test 1**

****

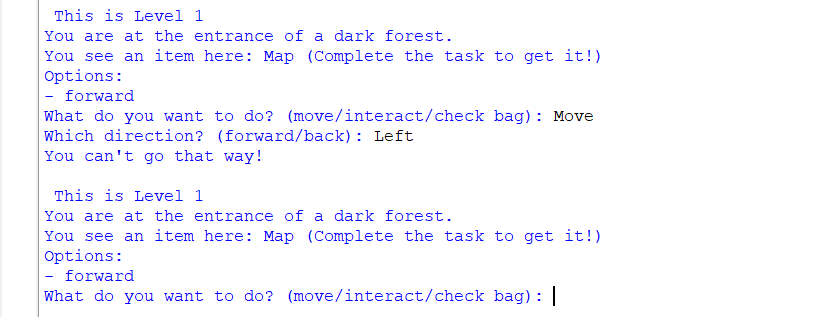
1. **Test 2**

****

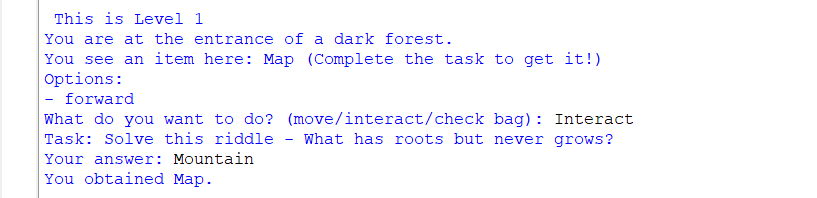
1. **Test 3**

****

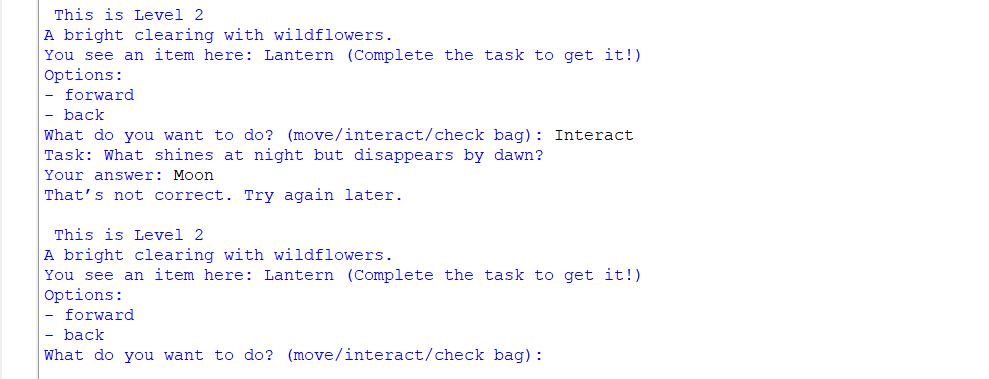
1. **Test 4**

****

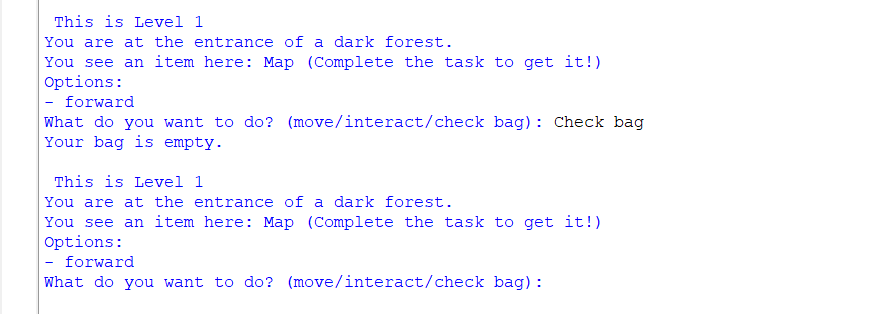
1. **Test 5**

****

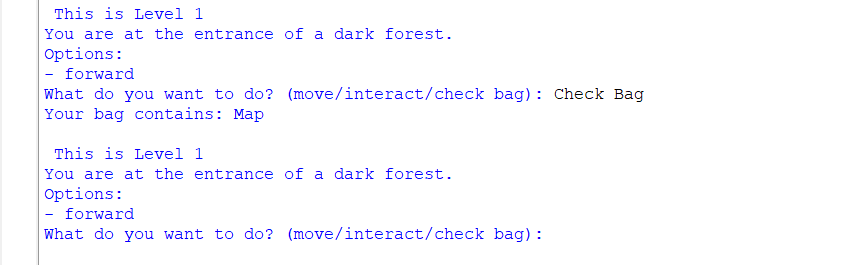
1. **Test 6**

****

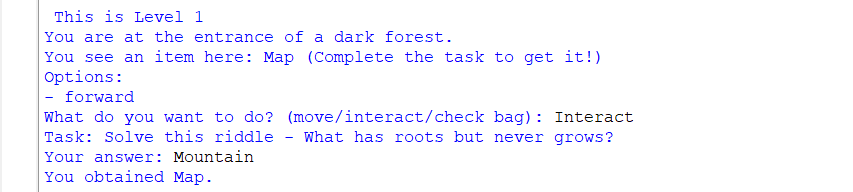
1. **Test 7**

****

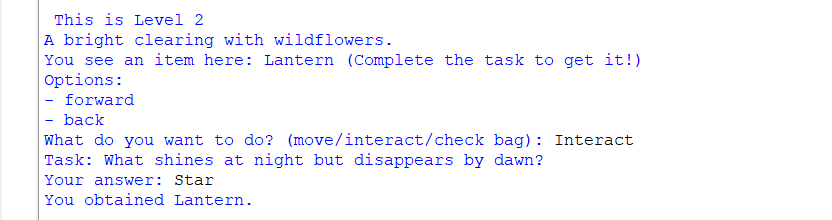
1. **Test 8**

****

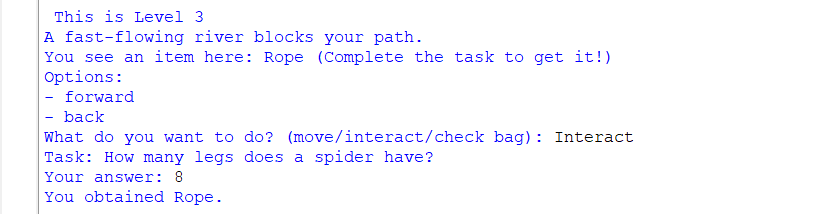
1. **Test 9**

****

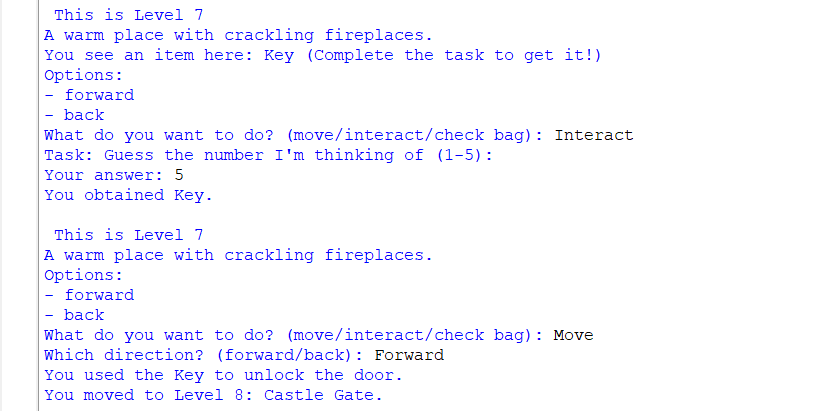
1. **Test 10**

****

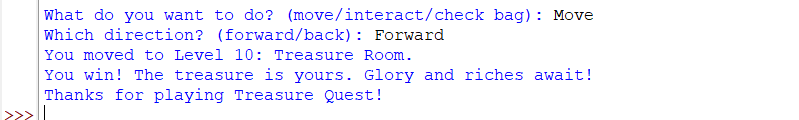
1. **Test 11**

****

1. **Test 12**

****

1. **Test 13**

****

**Testing against success criteria**

* Player name input works
* Game shows current position and choices
* Movement across all 10 levels
* Items can picked only after solving tasks
* Locked doors unlock with key
* Win message appears at treasure room

Task 04 – Technical Documentation

**Installation Instructions**

* Download the latest version of Python suitable for your operating system
* Save the provided game as treasure\_quest.py
* Open command prompt or terminal
* Use cd to off to the file’s folder
* Run the game

**Game Purpose**

The player must explore different areas, collect necessary items, solve puzzles, unlock gates, and reach the Treasure Room to win.

**Basic Gameplay Instructions**

* Move - Move to a different position. Player must specify a direction according to available options
* Interact - Interact with the current location to solve a task (like a riddle) and collect an item if successful
* Check bag - View the items currently collected and stored in the bag.

**Important**

* Bag can hold a maximum of 4 items.
* Some paths are locked and require a Key to unlock.
* You must solve riddles/tasks correctly to receive valuable items.

**Game Guide**

* Solve puzzles presented at each level.
* Collect key items after solving the tasks.
* Use the Key obtained at Level 7 (Village Inn) to unlock the Castle Gate.
* Continue moving until reaching the Treasure Room at Level 10.
* Successfully reach Level 10: Treasure Room.
* A special congratulations message will appear, indicating that you have completed the game.

**System Design Overview**

* The game world is created using a Python dictionary where each level is a key.
* The Player object tracks the player’s name, current position, and bag.
* The movement system uses available options inside each level’s dictionary entry.
* Task solving is linked to earning items.
* Victory occurs when reaching Level 10 after unlocking the required paths

## **Identification and Purpose of Subprograms**

1. **main()**

Purpose: Controls the game loop. Start the game, Asks for the player’s name and continuously asks for player actions (Move, Interact, Check bag) until the game ends.

1. **display\_world(world, player)**

Purpose: Displays the description of the current level the player is in, as well as available movement options and any item or task if present.

1. **move(world, player, direction)**

Purpose: Handles all movement logic. Checks if the chosen direction is valid, updates the player’s position, and checks for locked paths (E.g.: Castle gate).

1. **interact(world, player)**

Purpose: Allows the player to interact with the current level. If there’s a task. It prompts the player to solve it. If successful, the item is added to the player’s bag.

1. **get\_level\_number(position)**

Purpose: Extracts and returns the level number from a level name string (e.g., “Level 3: Riverbank” 🡪 “Level 3”). Used to display level headers.

1. **show\_levels(player)**

Purpose: Visually shows the level map or progress list with a marker showing the player’s current position.

1. **Player (Class)**

Purpose: Represents the player. Stores their name, current position, and the bag.