Initial Setup

The development environment for this project will be set up on a windwos desktop running windows, utilizing Visual Studio Code as the Integrated Development Environment (IDE). Instead of C++, this project will be implemented in Python.

Project Overview

The Adventure Game is a text-based interactive adventure where players explore different rooms, collect items, and use them to progress through the game. The game is composed of three rooms: the Start Room, the Hallway, and the Kitchen. Each room is represented by an ASCII art box, providing a visual representation of the room and its contents.

Rules of Text adventure game: The rules of Text adventure game implemented in this project will adhere to the standard rules, including:

Exploration: Players navigate through different rooms by using directional commands such as "go north", "go south", etc.

Item Collection: Players can pick up items found in each room using the command "take <item>". Items can be stored in the player's inventory for later use.

Item Usage: Players can use items in their inventory or within the room by typing "use <item>". This action can trigger events, solve puzzles, or unlock new areas.

Room Exits: Each room has exits leading to other rooms. Players can explore these exits by typing "go <direction>".

Room Descriptions: Upon entering a room, the game provides a description of the room, including any items present and the available exits.

Inventory Management: Players can view their inventory at any time to see which items they are currently carrying.

Game Loop: The game continues until the player decides to quit by typing "quit". Otherwise, the player can keep exploring, collecting items.

Help Command: Typing "help" displays a list of available commands and instructions on how to play the game.

Interaction Feedback: The game provides feedback to the player after each action, confirming the success or failure of the action and updating the game state accordingly.

Visual Representation: Rooms are visually represented using ASCII art, providing a unique and immersive experience for the player.

Requirements Analysis for Text Adventure Project:

User Interface:

1. The system should have a command-line interface (CLI) for user interaction.
2. Players should be able to input commands to interact with the game.

Game Initialization:

1. The system should allow players to start a new game.
2. Players should have the option to play solo or with multiple players.

Player Interaction:

1. Each player should be able to input commands during their turn.

2. Commands should include actions like moving between rooms, taking items, and using items.

Game Mechanics:

1. Players should navigate between different rooms using directional commands (e.g., north, south).
2. Each room should contain items that players can interact with.
3. Players should be able to pick up items found in rooms.
4. Items should have uses and effects when used by the player.

Player Management:

1. Players should be able to input their names at the start of the game.
2. The system should support multiple players, each with their own inventory.

Game Progression:

1. The game should determine win conditions for each player.
2. Win conditions might include finding a specific item, reaching a certain room, or solving a puzzle.

Game State Management:

1. The system should keep track of the current room for each player.
2. Inventory items for each player should be managed and tracked.
3. Used items should be recorded and updated accordingly.

Feedback and Output:

1. The system should provide descriptive feedback after each player action.
2. Feedback should include information on the current room, available exits, and any items found or used.

Help and Guidance:

1. Players should have access to a help command that provides instructions and available commands.
2. The system should guide players through the game's objectives and mechanics.

Scalability and Extensibility:

1. The system should be easily extensible to add new rooms, items, and game mechanics.
2. It should support future updates and expansions without major modifications to the codebase.

Behavior Driven Development (Gherkin Specifications)

Feature: Exploring Rooms in Adventure Game
AS A CLI USER/PLAYER: I want to enter player names So that each player can be identified in the game
Scenario: Player starts the game Given the game has started When the game initializes Then the game displays the current room with its description
Scenario: Player moves to a valid adjacent room
Given the player is in the start room **When** the player chooses to go north **Then** the player moves to the hallway room And the game displays the hallway room with its description
Scenario: Player moves to an invalid room
Given the player is in the kitchen room **When** the player chooses to go east **Then** the game displays a message that there is no room in that direction
Scenario: Player picks up an item
Given the player is in the start room And there is a key in the start room **When** the player chooses to take the key **Then** the player picks up the key And the game displays the start room without the key
Scenario: Player tries to pick up a non-existent item
Given the player is in the hallway room When the player tries to take a sword Then the game displays a message that there is no sword here

Scenario: Player uses an item from inventory
Given the player has a key in their inventory And the player is in the hallway room When the player chooses to use the key Then the game displays a message that the player used the key
Scenario: Player uses an item from the room
Given the player is in the kitchen room And there is a knife in the kitchen room When the player chooses to use the knife Then the game displays a message that the player used the knife
Scenario: Player tries to use a non-existent item
Given the player is in the start room When the player tries to use a hammer Then the game displays a message that the player doesn't have a hammer
Scenario: Player tries to use a non-existent item
Given the player is in the start room When the player enters a command "jump" Then the game displays a message that the command is invalid
Scenario: Player enters an invalid command
Given the player is in the start room When the player enters a command "jump" Then the game displays a message that the command is invalid
Scenario: Player requests help
Given the player is in the game When the player types "help" command Then the game displays a list of available commands and their descriptions
Scenario: Player quits the game
Given the player is in the game When the player chooses to quit

Then the game displays a farewell message
Scenario: Player tries to move without specifying direction
Given the player is in the hallway room When the player tries to go without specifying direction Then the game displays a message that the direction is missing

Data Model

Input:

- User Input (Standard Input)

Output Message:

- Welcome to the adventure game!
- Type 'help' for instructions.
- Current Room: [Room Name]
- [Room ASCII Art]
- Description: [Room Description]
- Exits: [Available Exits]
- You picked up the [item_name]
- There is no [item_name] here.
- You used the [item_name]
- You don't have a [item_name]
- Invalid command. Type 'help' for instructions.
- Available commands: go <direction>, take <item>, use <item>, help, quit
- Good luck!

Error:

- Error Values
    - Invalid argument
- Error Message

- Exit Code

Room Model:

Each room in the game is represented by a Room object.

Attributes:

name: Name of the room (string).

description: Description of the room (string).

exits: Dictionary representing exits from the room, where keys are directions (strings) and values are names of adjacent rooms (strings).

items: List of items present in the room (list of strings).

used_items: List of items already used in the room (list of strings).

Item Model:

Items within the game are represented by strings.

Player Model:

Each player in the game is represented by a Player object.

Attributes:

inventory: List of items the player is carrying (list of strings).

Game Model:

The game itself is represented by a Game object.

Attributes:

player: Instance of the Player class.

current_room: Current room the player is in (instance of Room).

rooms: Dictionary mapping room names to Room objects.

Input Model:

Players interact with the game by providing input, which is a sequence of characters.

Input: seq<char>

Output Message Model:

Output messages generated by the game during gameplay.

Output_Message: {Message1, Message2, ..., MessageN}

For example:

Output_Message = {"Welcome to the adventure game!", "Type 'help' for instructions.", "Invalid command. Type 'help' for instructions.", ...}

Game Command Model:

Commands available to the player during the game.

Command: seq<string>

For example:

Command = {"go", "take", "use", "help", "quit"}

Axiomatic Definitions and Functions

Axiomatic Definitions:

Room:

A room represents a location within the game environment.

Each room has:

A name: Identifies the room.

A description: Provides information about the room.

Exits: Defines the possible directions the player can move from the room.

Items: Objects present in the room that the player can interact with.

Used items: Items that the player has interacted with in the room.

Player:

A player is the character controlled by the user.

Each player has:

An inventory: A collection of items the player is carrying.

Game:

The game manages the flow of the game and interactions between the player and the game environment.

It keeps track of:

- The current room: The room where the player currently is.
- Collection of rooms: All the rooms available in the game.

Functions:

display_info (Room):

Displays the details of the current room, including ASCII art representation, description, and available exits.

add_item (Player):

Adds an item to the player's inventory.

remove_item (Player):

Removes an item from the player's inventory.

add_room (Game):

Adds a new room to the game.

start_game (Game):

Initializes the game by selecting a random room as the starting point and begins gameplay.

display_current_room (Game):

Displays information about the current room.

move (Game):

Moves the player to a new room based on the given direction.

take_item (Game):

Allows the player to take an item from the current room and add it to their inventory.

use_item (Game):

Allows the player to use an item from their inventory or the current room.

play (Game):

Main loop of the game where the player can input commands.

Handles movement, item interaction, and quitting the game.

T2 Implementation

Implementation of the program can now begin, considering two types of functions: pure and impure. Pure functions don't modify the program state outside their scope, while impure functions do. Additionally, there are totalised and non-totalised functions. Totalised functions cover all possible input-value pairs, while non-totalised functions don't. It's preferable to create pure, totalised functions wherever feasible.

Class: Room

Description: Represents a room in the game environment.

```
class Room:
    def __init__(self, name, description, exits, items=None):
        self.name = name
        self.description = description
        self.exits = exits
        self.items = items if items else []
        self.used_items = []
```

Function: display_info()

```
def display_info(self):
        # Define the ASCII art for each room
```

```
        room_art = {
            "start": [
                "+-------------------------------------------------------+",
                "|    Start room                                         |",
                "|                                                       |",
                "|   Items:                                              |",
                "|    -" + "\n|   - ".join(self.items).center(17) + "    ",
                "|                                                       |",
                "|   Used Items:                                         |",
                "|    -" + "\n|   - ".join(self.used_items).center(17) +"",
                "|                                                       |",
                "+-------------------------------------------------------+"
            ],
            "hallway": [
                "+-------------------------------------------------------+",
                "|   Hallway                                             |",
                "|                                                       |",
                "|   Items:                                              |",
                "|    -" + "\n|   - ".join(self.items).center(17) + "   ",
                "|                                                       |",
                "|   Used Items:                                         |",
                "|    -" + "\n|   - ".join(self.used_items).center(17)+"",
                "|                                                       |",
                "+-------------------------------------------------------+"
            ],
            "kitchen": [
                "+-------------------------------------------------------+",
                "|    Kitchen                                            |",
                "|                                                       |",
                "|   Items:                                              |",
                "|    -" + "\n|   - ".join(self.items).center(17) + "  ",
                "|                                                       |",
                "|   Used Items:                                         |",
                "|    -" + "\n|   - ".join(self.used_items).center(17) +"",
                "|                                                       |",
                "+-------------------------------------------------------+"
            ]
        }

        # Print the room's ASCII art
        print("\n".join(room_art[self.name]))
```

```
        # Print room description and exits
        print("Description:", self.description)
        print("Exits:", ", ".join(self.exits.keys()))
```

Description: Displays information about the current room, including ASCII art representation, description, and available exits.

Purity: Non-pure (displays output).

Totality: Totalized (always produces output).

Class: Player

Description: Represents the player in the game.

Attributes:

- inventory: List of items the player is carrying.

```
class Player:
 def __init__(self):
       self.inventory = []
```

add_item(item): Adds an item to the player's inventory.

```
  def add_item(self, item):
       self.inventory.append(item)
```

Purity: Impure (modifies state).

Totality: Totalized (always adds an item).

remove_item(item): Removes an item from the player's inventory.

```
  def remove_item(self, item):
       if item in self.inventory:
           self.inventory.remove(item)
           return item
       else:
           return None
```

Purity: Impure (modifies state).

Totality: Non-totalized (may not remove an item if it's not present).

Class: Game

```
class Game:
    def __init__(self):
        self.player = Player()
        self.current_room = None
        self.rooms = {}
```

Description: Orchestrates the game flow.

Attributes:

- player: Instance of the Player class.
- current_room: Current room where the player is.
- rooms: Dictionary containing all the rooms in the game.

Functions:

add_room(room): Adds a room to the game.

```
def add_room(self, room):
      self.rooms[room.name] = room
```

Purity: Impure (modifies state).

Totality: Totalized (always adds a room).

start_game(): Starts the game by selecting a random room and initiating gameplay.

```
 def start_game(self):
       self.current_room = random.choice(list(self.rooms.values()))
       self.play()
```

Purity: Impure (modifies state).

Totality: Totalized (always starts the game).

display_current_room(): Displays information about the current room.

```
  def display_current_room(self):
       print("\nCurrent Room:", self.current_room)
       self.current_room.display_info()
```

Purity: Non-pure (displays output).

Totality: Totalized (always displays the current room).

move(direction): Moves the player to another room based on the given direction.

```
def move(self, direction):
       if direction in self.current_room.exits:
           next_room_name = self.current_room.exits[direction]
           if next_room_name in self.rooms:
               self.current_room = self.rooms[next_room_name]
               self.display_current_room()  # Update display
```

```
            else:
                print("There is no room in that direction.")
        else:
            print("You cannot go in that direction.")
```

Purity: Impure (modifies state).

Totality: Totalized (always moves the player).

take_item(item_name): Allows the player to take an item from the current room and add it to their inventory.

```
  def take_item(self, item_name):
        if item_name in self.current_room.items:
            item = self.current_room.items.remove(item_name)
            self.player.add_item(item)
            print("You picked up the", item_name)
            self.display_current_room()  # Update display
        else:
            print("There is no", item_name, "here.")
```

Purity: Impure (modifies state).

Totality: Totalized (always allows the player to take an item).

use_item(item_name): Allows the player to use an item from their inventory or the current room.

```
 def use_item(self, item_name):
        if item_name in self.player.inventory:
            print("You used the", item_name)
            self.current_room.used_items.append(item_name)  # Add the used item to
used_items list
        elif item_name in self.current_room.items:
            print("You used the", item_name)
            self.current_room.used_items.append(item_name)  # Add the used item to
used_items list
        else:
            print("You don't have a", item_name)
        self.display_current_room()  # Update display
```

Purity: Impure (modifies state).

Totality: Totalized (always allows the player to use an item).

play(): Main loop of the game where the player can input commands.

```
def play(self):
        print("Welcome to the adventure game!")
        print("Type 'help' for instructions.")
        self.display_current_room()  # Initial display

        while True:
            command = input("What would you like to do? ").strip().lower().split()
```

```
            if command[0] == 'go' and len(command) > 1:
                self.move(command[1])
            elif command[0] == 'take' and len(command) > 1:
                self.take_item(command[1])
            elif command[0] == 'use' and len(command) > 1:
                self.use_item(command[1])
            elif command[0] == 'help':
                print("Available commands: go <direction>, take <item>, use <item>,
help, quit")
            elif command[0] == 'quit':
                print("Good luck!")
                break  # Stop the game
            else:
                print("Invalid command. Type 'help' for instructions.")
```

Purity: Impure (handles I/O).

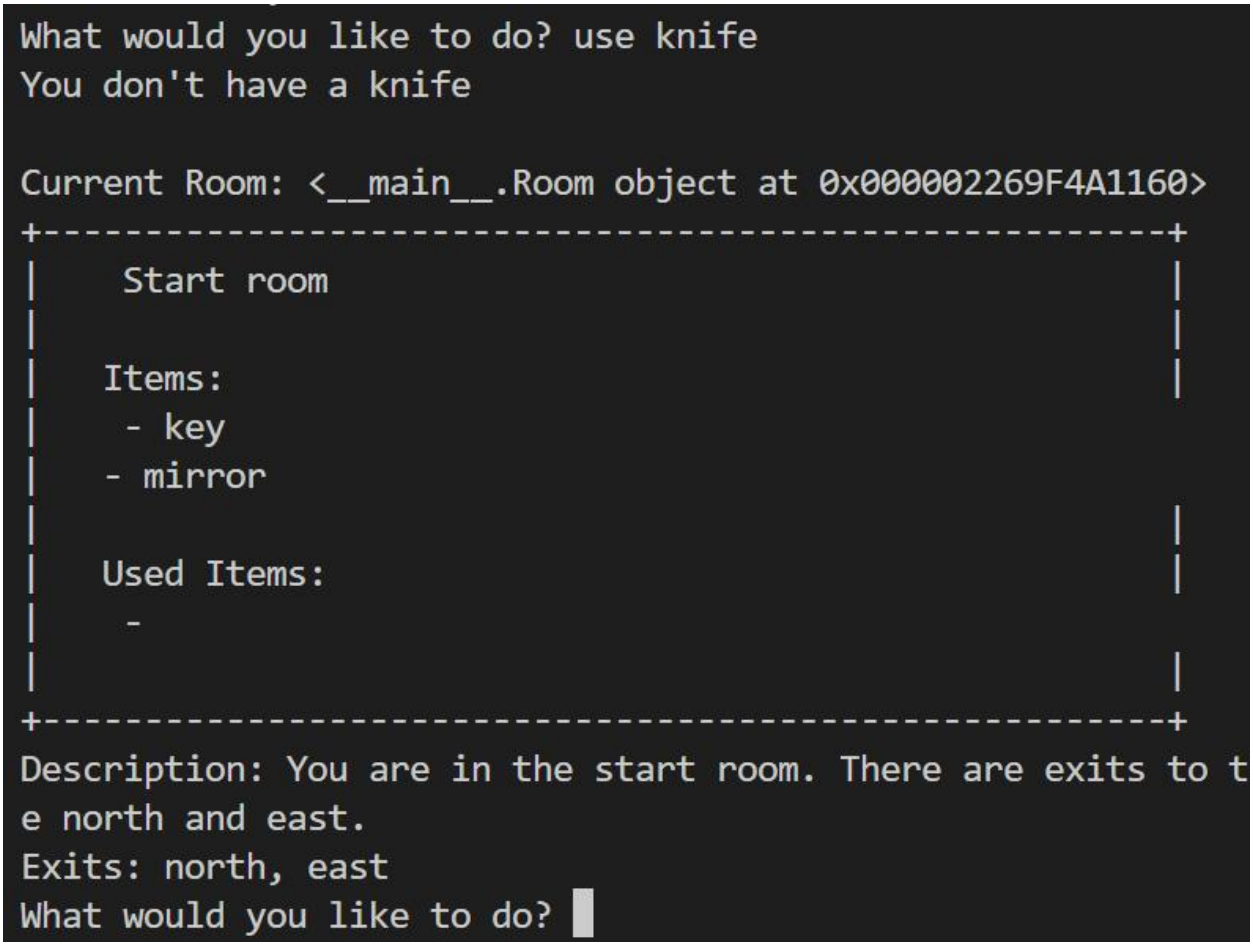
Totality: Totalized (always runs the game loop).

Output

```
Welcome to the adventure game!
Type 'help' for instructions.

Current Room: <__main__.Room object at 0x000002269F4A2E70>
+-----------------------------------------------------+
|    Kitchen                                          |
|                                                     |
|   Items:                                            |
|    -knife
|   - fruit
|   - nuts
|                                                     |
|   Used Items:                                       |
|    -
|                                                     |
+-----------------------------------------------------+
Description: You are in the kitchen room. There's an exit to
the west.
Exits: west
What would you like to do?
```

```
What would you like to do? use fruit
You used the fruit

Current Room: <__main__.Room object at 0x000002269F4A2E70>
+-----------------------------------------------------+
|    Kitchen                                          |
|                                                     |
|   Items:                                            |
|    -knife
|   - fruit
|   - nuts
|                                                     |
|   Used Items:                                       |
|    -      fruit
|                                                     |
+-----------------------------------------------------+
Description: You are in the kitchen room. There's an exit to
the west.
Exits: west
What would you like to do?
```

```
What would you like to do? go west

Current Room: <__main__.Room object at 0x000002269F4A1160>
+-----------------------------------------------------------+
|    Start room                                             |
|                                                           |
|   Items:                                                  |
|    - key                                                  |
|   - mirror                                                |
|                                                           |
|   Used Items:                                             |
|    -                                                      |
|                                                           |
+-----------------------------------------------------------+
Description: You are in the start room. There are exits to th
e north and east.
Exits: north, east
```

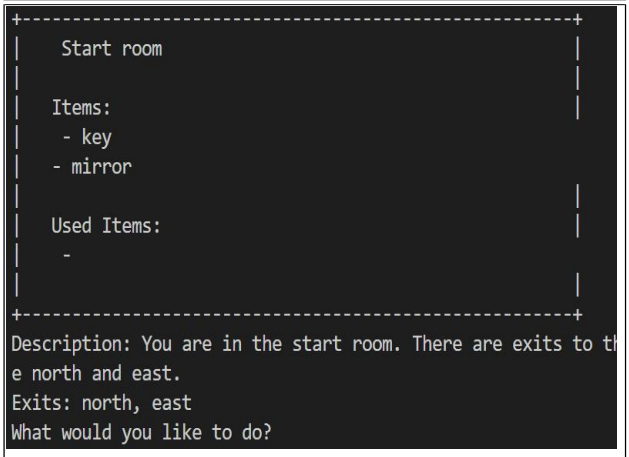


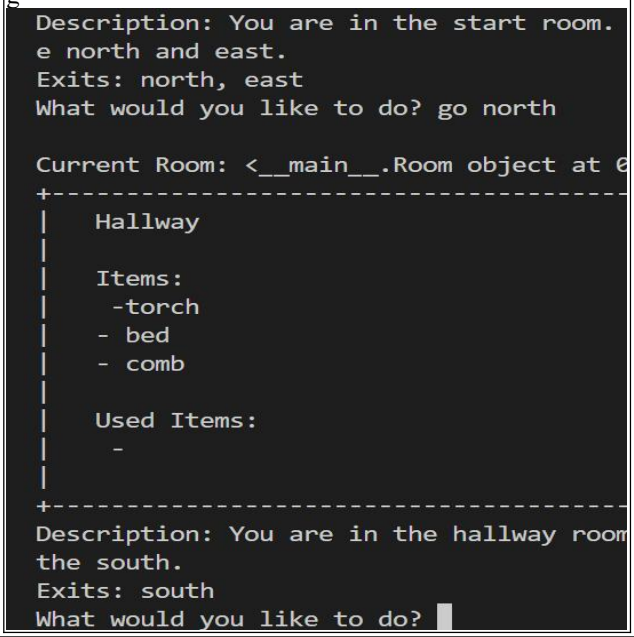
Testing

Both manual and automated tests are necessary to ensure that the features work as expected according to the Gherkin specifications and other planning components. Manual testing involves using the software implementation to assess whether expected outputs are returned for specific user inputs. Lets start one by one.

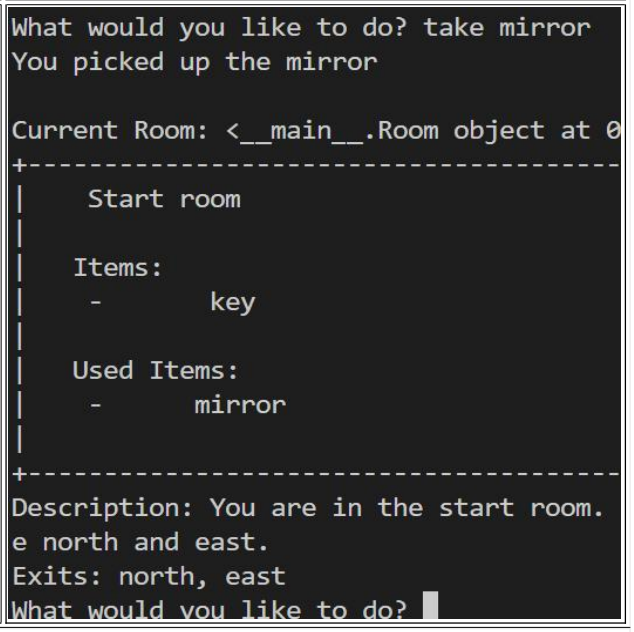
Manual Testing

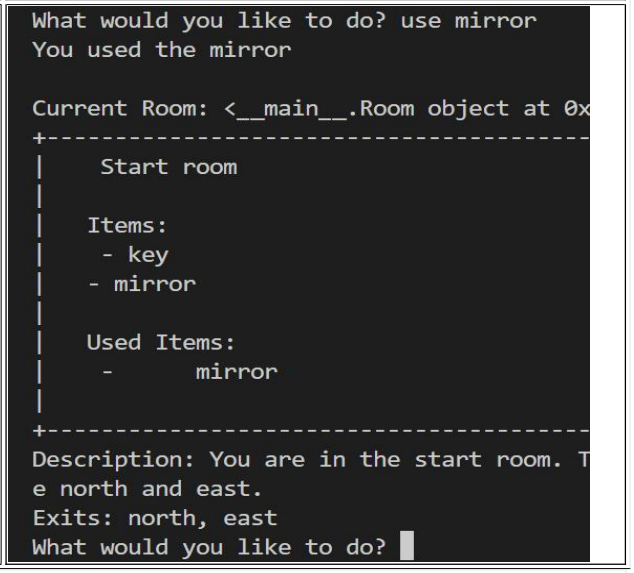
Test Case ID	1	Passed
Software Feature to Test	Displaying room information	
Steps to Do	Start the game	Display the current room

Test Case ID	1	Passed
Expected Output	Start room information displayed	

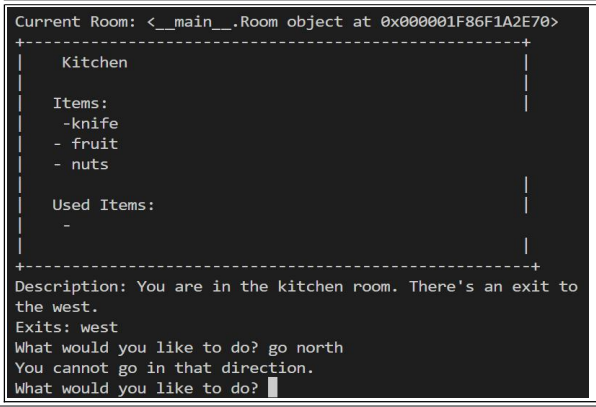
Test Case ID	2	Passed/Failed
Software Feature to Test	Moving to another room	
Steps to Do	Move to the hallway	Display the current room
Expected Output	Hallway room information displayed	g 

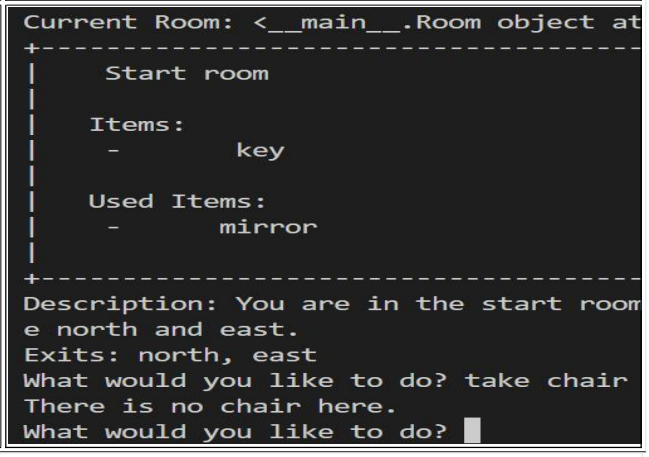
Test Case ID	3	Passed/Failed
Software Feature to Test	Taking an item	
Steps to Do	Take the mirror from the start room	Display the current room

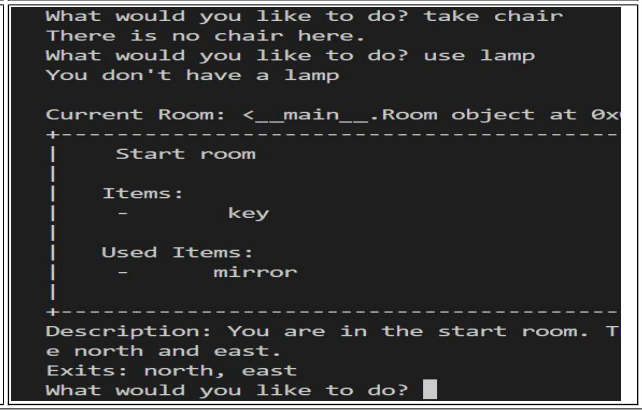
Test Case ID	3	Passed/Failed
Expected Output	Mirror taken and Start room information displayed	

Test Case ID	4	Passed/Failed
Software Feature to Test	Using an item	
Steps to Do	Use the mirror	Display the current room
Expected Output	Mirror used and Start room information displayed	

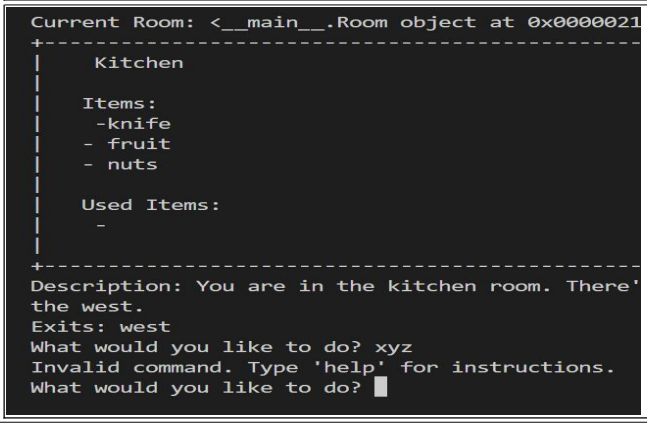
Test Case ID	5	Passed/Failed
Software Feature to Test	Attempting to move to a non-existent room	
Steps to Do	Try to move north from the kitchen	Display error message

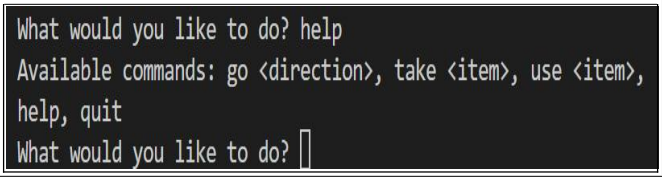
Test Case ID	5	Passed/Failed
Expected Output	Error message displayed	

Test Case ID	6	Passed/Failed
Software Feature to Test	Attempting to take a non-existent item	
Steps to Do	Try to take a chair from the start room	Display error message
Expected Output	Error message displayed	

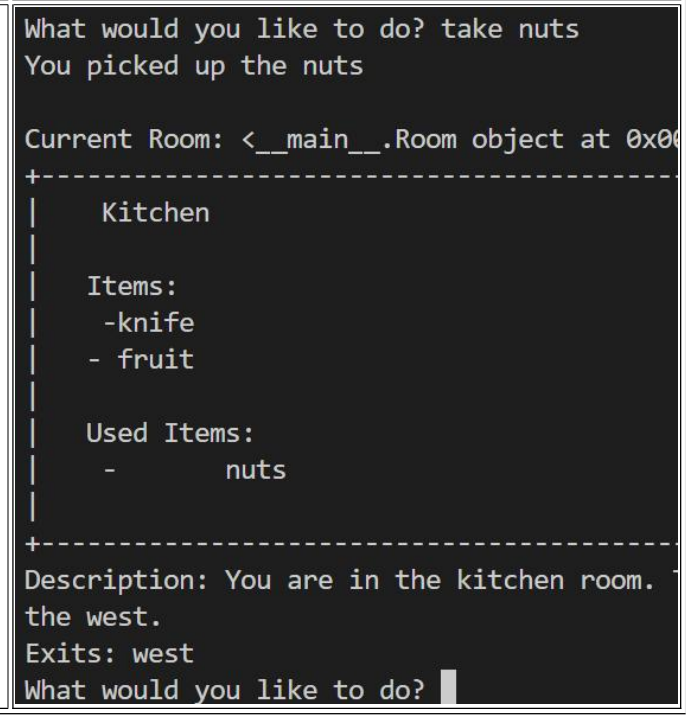
Test Case ID	7	Passed/Failed
Software Feature to Test	Attempting to use a non-existent item	
Steps to Do	Try to use a lamp	Display error message
Expected Output	Error message displayed	

Test Case ID	8	Passed/Failed
Software Feature to Test	Quitting the game	
Steps to Do	Type 'quit'	Display farewell message
Expected Output	Farewell message displayed	

Test Case ID	9	Passed/Failed
Software Feature to Test	Typing an invalid command	
Steps to Do	Type 'xyz'	Display error message
Expected Output	Error message displayed	

Test Case ID	10	Passed/Failed
Software Feature to Test	Typing 'help' for instructions	
Steps to Do	Type 'help'	Display available commands
Expected Output	Available commands displayed	

Test Case ID	11	Passed/Failed
Software Feature to Test	Taking an item from an empty room	
Steps to Do	Take an item from the kitchen	Display item used message

Test Case ID	11	Passed/Failed
Expected Output	Display item used	

Automated testing:

For automated testing of frakle “unittest” library is best and using it.

The unittest library in Python is a built-in testing framework that allows you to write test cases for your code in a structured and organized manner. It provides a set of tools for constructing and running tests, as well as making assertions about the behavior of your code.

Setup for testing

```
class TestAdventureGame(unittest.TestCase):
    def setUp(self):
        # Create rooms
        rooms_data = {
            "start": {"description": "You are in the start room. There are exits to the north and east.", "exits"
            "hallway": {"description": "You are in the hallway room. There's an exit to the south.", "exits": {"s
            "kitchen": {"description": "You are in the kitchen room. There's an exit to the west.", "exits": {"we
        }

        self.rooms = {name: Room(name, **data) for name, data in rooms_data.items()}
        self.game = Game()

        for room in self.rooms.values():
            self.game.add_room(room)

    @patch('builtins.input', side_effect=['go north', 'go east', 'take key', 'take chair', 'use key', 'use chair'
    @patch('sys.stdout', new_callable=StringIO)
```

Testing the game code

```
def test_game(self, mock_stdout, mock_input):
    self.game.start_game()
    output = mock_stdout.getvalue()
    expected_output = [
        "Current Room: start",
        "Welcome to the adventure game!",
        "Type 'help' for instructions.",
        "Description: You are in the start room. There are exits to the north and east.",
        "Exits: north, east",
        "What would you like to do? ",
        "Current Room: hallway",
        "Description: You are in the hallway room. There's an exit to the south.",
        "Exits: south",
        "What would you like to do? ",
        "Current Room: kitchen",
        "Description: You are in the kitchen room. There's an exit to the west.",
        "Exits: west",
        "What would you like to do? ",
        "You picked up the key",
        "Current Room: start",
        "Description: You are in the start room. There are exits to the north and east.",
```

```
def test_game(self, mock_stdout, mock_input):
        "Description: You are in the start room. There are exits to the north and east.",
        "Exits: north, east",
        "What would you like to do? ",
        "Current Room: kitchen",
        "Description: You are in the kitchen room. There's an exit to the west.",
        "Exits: west",
        "What would you like to do? ",
        "Available commands: go <direction>, take <item>, use <item>, help, quit",
        "Current Room: kitchen",
        "Description: You are in the kitchen room. There's an exit to the west.",
        "Exits: west",
        "What would you like to do? ",
        "Good luck!\n"
    ]

    output_lines = output.split('\n')
    expected_output_lines = [line.strip() for line in expected_output]

    for expected_line, actual_line in zip(expected_output_lines, output_lines):
        self.assertEqual(expected_line, actual_line)
```

Running the code and output of tests:

```
Current Room: <text_adventure.Room object at 0x000001C9D28905F0>
+-------------------------------------------------------+
|   Hallway                                             |
|                                                       |
|   Items:                                              |
|    -torch
|   - bed
|   - comb
|                                                       |
|   Used Items:                                         |
|    -
|                                                       |
+-------------------------------------------------------+
Description: You are in the hallway room. There's an exit to the south.
Exits: south
What would you like to do? take torch
You picked up the torch
```

```
What would you like to do? use comb
You used the comb

Current Room: <text_adventure.Room object at 0x000001C9D28905F0>
+-------------------------------------------------------+
|   Hallway                                             |
|                                                       |
|   Items:                                              |
|    -  bed
|   - comb
|                                                       |
|   Used Items:                                         |
|    -       comb
|                                                       |
+-------------------------------------------------------+
Description: You are in the hallway room. There's an exit to the south.
Exits: south
What would you like to do?
```

```
Description: You are in the hallway room. There's an exit to the south.
Exits: south
What would you like to do? go north
You cannot go in that direction.
What would you like to do?
```

```
What would you like to do? quit
Good luck!
```

Game is stopped when user quit it.

Appendix A

Full automated testing code:

```python
import unittest
from unittest.mock import patch
from io import StringIO
from text_adventure import Room, Player, Game

class TestAdventureGame(unittest.TestCase):
    def setUp(self):
        # Create rooms
        rooms_data = {
            "start": {"description": "You are in the start room. There are exits to the north and east.", "exits"
            "hallway": {"description": "You are in the hallway room. There's an exit to the south.", "exits": {"s
            "kitchen": {"description": "You are in the kitchen room. There's an exit to the west.", "exits": {"we
        }

        self.rooms = {name: Room(name, **data) for name, data in rooms_data.items()}
        self.game = Game()

        for room in self.rooms.values():
            self.game.add_room(room)

    @patch('builtins.input', side_effect=['go north', 'go east', 'take key', 'take chair', 'use key', 'use chair'
    @patch('sys.stdout', new_callable=StringIO)
    def test_game(self, mock_stdout, mock_input):
```

```
def test_game(self, mock_stdout, mock_input):
    output = mock_stdout.getvalue()
    expected_output = [
        "Current Room: start",
        "Welcome to the adventure game!",
        "Type 'help' for instructions.",
        "Description: You are in the start room. There are exits to the north and east.",
        "Exits: north, east",
        "What would you like to do? ",
        "Current Room: hallway",
        "Description: You are in the hallway room. There's an exit to the south.",
        "Exits: south",
        "What would you like to do? ",
        "Current Room: kitchen",
        "Description: You are in the kitchen room. There's an exit to the west.",
        "Exits: west",
        "What would you like to do? ",
        "You picked up the key",
        "Current Room: start",
        "Description: You are in the start room. There are exits to the north and east.",
        "Exits: north, east",
        "What would you like to do? ",
```

```
    def test_game(self, mock_stdout, mock_input):
            "What would you like to do? ",
            "Current Room: kitchen",
            "Description: You are in the kitchen room. There's an exit to the west.",
            "Exits: west",
            "What would you like to do? ",
            "Available commands: go <direction>, take <item>, use <item>, help, quit",
            "Current Room: kitchen",
            "Description: You are in the kitchen room. There's an exit to the west.",
            "Exits: west",
            "What would you like to do? ",
            "Good luck!\n"
        ]

        output_lines = output.split('\n')
        expected_output_lines = [line.strip() for line in expected_output]

        for expected_line, actual_line in zip(expected_output_lines, output_lines):
            self.assertEqual(expected_line, actual_line)

if __name__ == '__main__':
    unittest.main()
```

T4 Git version control discussion

Documentation greatly benefits projects by providing a clear explanation of what a codebase does and

how it can be used (Meza, 2018). It also assists secondary developers in understanding rules, approaches, naming conventions, comments, etc., thereby promoting maintainable code.

Commits in Git serve as checkpoints in the project's history. Each commit captures the state of the codebase at a specific moment, enabling developers to revert back to previous versions if necessary. This functionality is invaluable for the Farkle project as it allows for easy tracking of changes and quick recovery from errors or unintended modifications.

Pushing commits to the Git repository is how changes made on a local machine are shared with others. It ensures that all team members have access to the latest version of the codebase. Similarly, pulling updates the local machine with changes made by others, ensuring everyone is working with the most recent version.

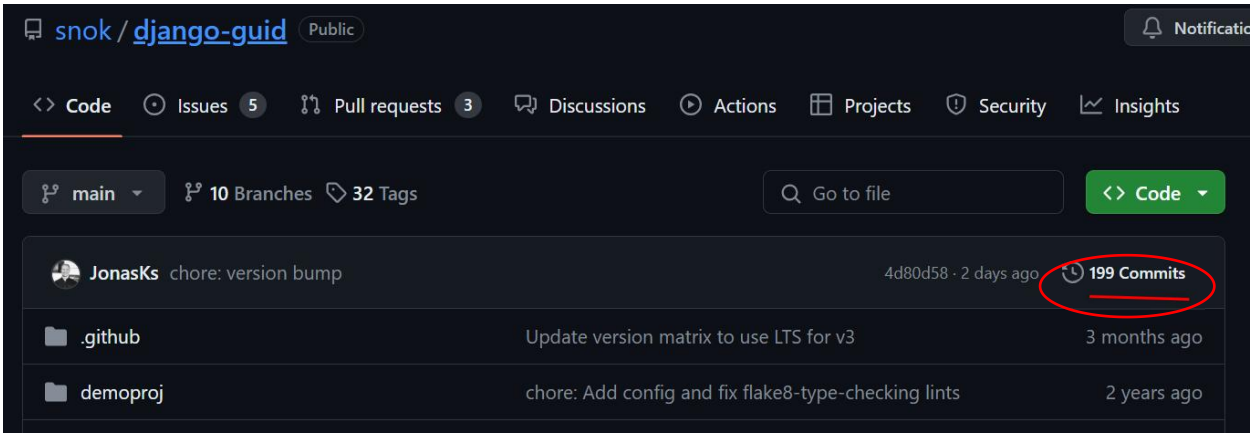
Pull requests are especially useful for team collaboration on the Farkle project. They allow developers to propose changes, have them reviewed by peers, and then merge them into the main repository. This process ensures that code changes are thoroughly examined before being integrated, maintaining code quality and stability.

Case study

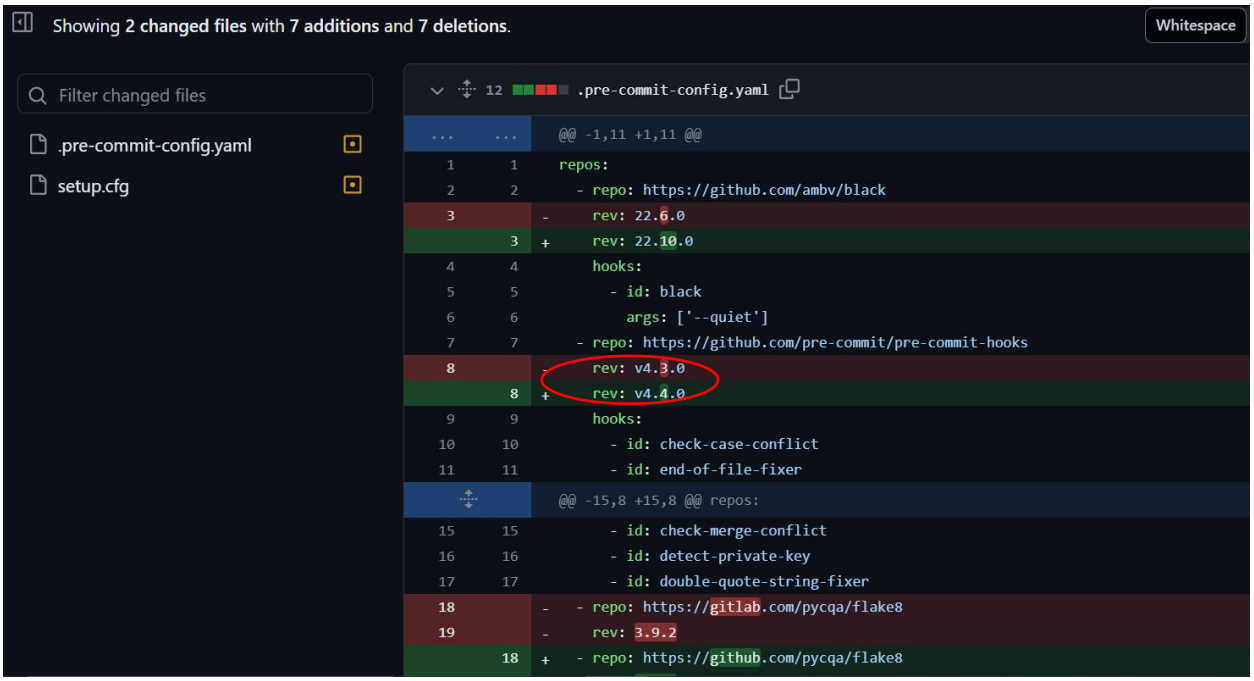
For a case study of git, I study python unittest help to do automated testing of my project.

https://github.com/reactjs/react-docgen

The commit history for kislyuk can be easily accessed by clicking the commits symbol when navigating GitHub, underlined in red in figure.



Commits are displayed in chronological order, with the latest commit appearing at the top of the list. By scrolling down or selecting "older," you can view earlier commits in the repository. Change in project code is highlited with red and green highliters as


Showing 2 changed files with 7 additions and 7 deletions.
Whitespace
Filter changed files
.pre-commit-config.yaml
setup.cfg
12 .pre-commit-config.yaml
@@ -1,11 +1,11 @@
repos:
- repo: https://github.com/ambv/black
- rev: 22.6.0
+ rev: 22.10.0
hooks:
- id: black
args: ['--quiet']
- repo: https://github.com/pre-commit/pre-commit-hooks
- rev: v4.3.0
+ rev: v4.4.0
hooks:
- id: check-case-conflict
- id: end-of-file-fixer
@@ -15,8 +15,8 @@ repos:
- id: check-merge-conflict
- id: detect-private-key
- id: double-quote-string-fixer
- - repo: https://gitlab.com/pycqa/flake8
- rev: 3.9.2
+ - repo: https://github.com/pycqa/flake8

Refrences:

The benefits and need of documentation of code:

https://iwconnect.com/the-benefits-of-code-documentation-why-its-a-must-have-for-your-development-team/ Accessed [28/04/2024]

Text adventure game guide:

https://en.wikipedia.org/wiki/Text-based_game Accessed [28/04/2024]

Git guid sample:

https://github.com/git-guides/git-clone Accessed [28/04/2024]