TEXT- BASED ADVENTURE GAME USING PYTHON

MOTION CUTS INTERNSHIP

PYTHON PROGRAMMING INTERNSHIP

WEEK 4

SUBMITTED BY:

BANUPRAKASH

COLLEGE NAME: DAYANANDA SAGAR COLLEGE OF ENGINEERING

CONTENTS

INTRODUCTION

Genre: Text-based adventure games are a genre of interactive fiction where the player progresses through the story by making choices through text commands.

Narrative Focus: The emphasis is on storytelling, with players engaging in a dynamic narrative where their decisions shape the outcome.

Imagination: Unlike graphical games, text-based adventures rely on the player's imagination to visualize the scenarios described in the text.

Choices Matter: Decision-making is a core element, impacting the plot, characters, and the overall progression of the game.

Command Interface: Players interact with the game by typing commands or selecting choices from predefined options, often in a command-line interface.

Puzzles and Challenges: These games often include puzzles or challenges that players must solve to advance, adding an element of problem-solving to the experience.

Exploration: Players explore virtual worlds through descriptive text, uncovering hidden locations, items, and story elements.

Character Interaction: Interaction with in-game characters is crucial, influencing relationships and affecting the unfolding narrative.

Multiple Endings: Many text-based adventures feature multiple endings, providing replay value based on different choices made throughout the game.

Nostalgia: While older in origin, text-based adventures remain popular, offering a nostalgic experience for gamers who appreciate the roots of interactive fiction.

ALGORITHM

1.Introduction:

- Display a welcome message and set the initial scene in a mysterious forest.
- Inform the player about the goal of reaching a magical castle.

2.Forest Path:

- Player enters a dark forest.
- Present choices to either investigate strange sounds or ignore them.
- If the player investigates:
- Encounter a friendly creature and receive a magical amulet.
- If the player ignores:
- Sounds were just the wind; continue the journey.

3. River Crossing:

The player encounters a wide river with a rickety bridge.

- Offer choices to either cross the bridge or find another way around.
- If the player crosses:

The bridge collapses, but the player manages to swim across.

- If the player finds another way:
- Discover a hidden path around the river.

4. Final Challenge:

The player reaches the magical castle.

- If the player has the magical amulet: - The amulet opens the castle doors, and the player successfully completes the journey.
- If the player doesn't have the amulet:
- The castle doors remain closed, and the journey ends here.

5. Error Handling: - Implement error handling for invalid inputs during the game, ensuring the program does not crash if the user provides unexpected input.

6. User Interface:

- Ensure a user-friendly interface by providing clear instructions at each decision point.
- Use a function to handle player choices and validate inputs.

7. Main Function:

- Execute the main function, which orchestrates the flow of the game by calling the introduction, forest_path, river_crossing, and final_challenge functions sequentially.

8. Run the Game:

- Check if the script is run directly (if __name__ == "__main__":) and execute

the main function.

PROGRAM

```
import time

def intro():
    print("Welcome to the Text Adventure Game!")
    time.sleep(1)
    print("You find yourself in a mysterious forest.")
    time.sleep(1)
    print("Your goal is to reach the castle at the end of the forest.")
    time.sleep(1)
    print("Be careful with your decisions. Good luck!\n")

def make_choice(choices):
    print("Choose your path:")
    for i, choice in enumerate(choices, start=1):
        print(f"{i}. {choice}")

    while True:
        try:
            user_input = int(input("Enter the number of your choice: "))
            if 1 <= user_input <= len(choices):
                return user_input
            else:
                print("Invalid input. Please enter a valid number.")
        except ValueError:
            print("Invalid input. Please enter a number.")

def forest_path():
print("You enter the dark forest.")
    time.sleep(1)
    print("As you walk deeper, you encounter a fork in the path.")

    choices = ["Take the left path.", "Take the right path."]
    user_choice = make_choice(choices)

    if user_choice == 1:
        print("You chose the left path.")
        time.sleep(1)
        print("You discover a hidden shortcut and make good progress.")
        return "shortcut"
    else:
        print("You chose the right path.")
        time.sleep(1)
        print("You encounter a group of hostile creatures.")
        choices = ["Fight them.", "Try to sneak past."]
        user_choice = make_choice(choices)

        if user_choice == 1:
            print("You bravely fight the creatures.")
            time.sleep(1)
            print("You defeat them and continue your journey.")
            return "fight"
        else:
            print("You attempt to sneak past the creatures.")
            time.sleep(1)
print("You manage to avoid them and proceed cautiously.")
            return "sneak"

def castle_path():
    print("You reach the castle gates.")
    time.sleep(1)
    print("The gates are locked, and you need to solve a riddle to enter.")

    riddle = "I speak without a mouth and hear without ears. I have no body, but I come alive with the wind. What am I?"
    print(f"\nRiddle: {riddle}")

    user_answer = input("\nEnter your answer: ").lower()

    if user_answer == "an echo":
        print("The gates open, and you enter the castle.")
        return "win"
    else:
        print("Incorrect answer. The gates remain closed.")
        return "lose"

def play_game():
    intro()
    path = forest_path()

    if path == "shortcut":
        print("You reach the castle gates faster using the shortcut.")
elif path == "fight":
        print("Despite the fight, you reach the castle gates.")
    elif path == "sneak":
        print("You successfully sneak past the creatures and reach the castle gates.")

    result = castle_path()

    if result == "win":
        print("Congratulations! You successfully reached the castle. You win!")
    else:
        print("Unfortunately, you couldn't enter the castle. Better luck next time!")

if _name_ == "_main_":
    play_game()
```

APPLICATIONS

1. Entertainment: The primary purpose is to entertain users by immersing them in a narrative-driven experience where they make choices that affect the outcome.

2. Educational Tools: They can be used for educational purposes, teaching critical thinking, decision-making, and problem-solving skills. They are also great for language learning.

3. Training Simulations: Text-based adventures can simulate real-world scenarios, offering a safe environment for training in areas like decision-making, crisis management, or customer service.

4. Storytelling Platforms: Authors and storytellers can use text-based games to create interactive narratives, engaging readers in a dynamic and immersive way.

5. Therapeutic Applications: In psychology, text-based games can be used for therapeutic purposes, such as exposure therapy or helping individuals cope with stress and anxiety.

6. Interactive Fiction for Marketing: Businesses can use text-based games for interactive marketing campaigns, engaging users in a story that promotes products or services.

7. Game Design Learning: Aspiring game developers can use text-based games to learn the fundamentals of game design, story crafting, and user engagement.

8. Cognitive Testing: Text-based adventures can be employed in psychological studies to assess cognitive abilities, decision patterns, and problem-solving skills.

9. Historical and Cultural Exploration: Games set in historical or cultural contexts can be both entertaining and educational, providing players with insights into different time periods or societies.

10. Community Building: Text-based games can foster community engagement, as players discuss strategies, share experiences, and collectively explore the game's world

ADVANTAGES

1. Imagination Stimulation: They encourage players to use their imagination to visualize the game world, characters, and scenarios, fostering creativity.

2. Low System Requirements: Text-based games are lightweight and don't require advanced graphics, making them accessible to a broader range of devices, including older computers or mobile devices.

3. Focus on Storytelling: With the absence of graphics, developers can focus more on crafting intricate and engaging storylines, leading to rich narrative experiences.

4. Accessible Gameplay: Text-based games often have straightforward controls, making them accessible to players of various ages and gaming backgrounds.

5. Reduced Development Complexity: Creating a text-based game is generally less complex than developing a graphical game, allowing for faster and more straightforward development cycles.

6. Educational Value: These games can enhance reading and comprehension skills, as players need to follow the story, make decisions, and solve problems through textual prompts.

7. Wide Player Agency: Text-based games often provide players with a high degree of agency, allowing them to make choices that significantly impact the game's outcome.

8. Easy to Expand and Modify: Developers can easily expand or modify text-based games by adding new storylines, characters, or scenarios without the need for extensive graphical assets.

9. Inclusive Design: Text-based games can be more inclusive for players with visual impairments since the gameplay is focused on text and doesn't rely on visual elements.

10. Nostalgia and Retro Appeal: For some players, text-based games evoke a sense of nostalgia, offering a retro gaming experience reminiscent of early computer games.

CONCLUSION

In summary, the creation of this Python-based text adventure game has been a rewarding journey, blending storytelling and programming to craft an interactive and captivating experience. The game successfully incorporates dynamic narratives that respond to user choices, showcasing a solid grasp of Python programming principles. Throughout the development process, challenges were confronted and conquered, contributing not only to improved coding proficiency but also enhanced problem-solving skills. Looking ahead, potential expansions, such as adding more story branches and incorporating advanced Python concepts, could elevate the project further. This endeavor stands as a testament to the progress made in programming and game development, offering valuable insights for future projects aimed at creating more sophisticated and engaging user experiences.