Project: Text-Based Adventure Game

Objective:

Create an interactive text-based adventure game where players make choices that influence the story's outcome. The game will involve branching paths, inventory management, and simple combat or puzzle-solving mechanics.

Concept and Story Design:

Objective:

- Develop the game's storyline, setting, and characters.

Tasks:

- Brainstorm and write a storyline with multiple paths. Create a flowchart showing different choices and their outcomes.

Example:

- A story set in a fantasy world, where choices lead to finding a treasure, encountering a dragon, etc.

Tips:

- Encourage creativity in storytelling.
- Use tools like pen and paper or digital flowchart software for mapping the story.
- Start by brainstorming the setting (e.g., fantasy world, space adventure, detective story).
- Define the main characters and their roles in the story.
- Outline the main plot and key events.
- Create a flowchart to visualize how players' choices lead to different story branches and endings. Tools like draw.io or Lucidchart can be helpful.
- Write a brief summary of each branch to guide your coding process.

Basic Game Structure and Player Input:

Objective:

- Set up the basic structure for displaying text and getting user input.

Tasks:

- Write Python functions to print story segments and to get choices from the user.

Tips:

- Familiarize yourself with the print() function for displaying text.
- Use the input() function to get choices from the player.

- Write a function show_story() to display a piece of the story.
- Create a function get_choice() to let players input their decisions.
- Let the user create their character in the beginning so they are inserted into the game
- Test these functions with sample text and choices to ensure they work correctly.

Implementing Story Branches

Objective:

- Create branching paths in the story based on user choices.

Tasks:

- Use if-elif-else statements to lead the player to different parts of the story.

Tips:

- Review if, elif, and else statements for decision-making.
- Link each choice in get_choice() to a corresponding story outcome using these statements.
- Start with one branch and expand to multiple, ensuring logical consistency.
- Test each branch by playing through these paths.

Adding Inventory and Items

Objective:

- Implement an inventory system where players can collect and use items.

Tasks:

- Create a list to store inventory items. Write functions to add and use items.

Tips:

- Learn about Python lists to manage the inventory.
- Create a global list, inventory, to store item names.
- Write add_to_inventory(item) to add items to this list.
- Implement use_item(item) to utilize and remove items from the inventory.
- Integrate these functions into your story where items are found or used.

Simple Combat or Puzzle System

Objective:

- Introduce a basic combat system or puzzles for player interaction.

Tasks:

- Implement simple logic for combat or solving puzzles.

Example:

- A basic combat system can involve random number generation to simulate attack and defense.

Tips:

- For combat, introduce a simple health system for the player and enemy.
- Use Python’s random module for attack outcomes.
- For puzzles, create logic-based challenges (e.g., riddles, sequence solving).
- Implement these systems at specific points in the story.
- Ensure combat and puzzles impact the game's progression (e.g., losing health, gaining items).

Expanding the Game World

Objective:

- Add more depth to the game with additional paths, characters, and locations.

Tasks:

- Develop and code additional storylines and scenarios.

Example:

- Adding a new character like a wizard who gives puzzles or a new location like a haunted forest.

Tips:

- Identify parts of the story that can be further developed.
- Add new characters, each with unique interactions and choices.
- Introduce new locations and describe them vividly.
- Ensure each new element enriches the story and integrates well with existing paths.
- Revisit your flowchart to incorporate these new elements.

Polishing and Refining

Objective:

- Test, debug, and refine the game.

Tasks:

- Playtest the game to find and fix bugs. Refine storylines and code based on feedback.

Example:

- Identifying and fixing a scenario where a certain path leads to an unexpected end.

Tips:

- Playtest the game, noting any bugs or logical inconsistencies.
- Review your code for potential errors or improvements.
- Refine story elements for clarity and engagement.
- Solicit feedback from peers to identify areas for improvement.
- Iterate on your design, refining the story and code.

Core Requirements

Game Narrative:

- Create a compelling storyline with a clear beginning, middle, and end.
- Include at least five significant locations or events that the player can explore.

Character Development:

- Implement a player character with at least three attributes (e.g., health, strength, mana) that impact gameplay.
- Include at least five non-player characters (NPCs) with whom the player can interact.

Choices and Consequences:

- Provide the player with meaningful choices that influence the story's outcome.
- Implement at least three branching paths based on player decisions, leading to different endings.

Inventory System:

- Create a simple inventory system where players can collect and use items that affect the game (e.g., keys to unlock doors, health potions).

Challenges:

- Incorporate at least two challenges: puzzles (e.g., solving a riddle) and obstacles (e.g., navigating a maze or overcoming an enemy).

User Interface:

- Design an intuitive text-based interface that clearly displays information to the player (e.g., current location, inventory contents) and prompts for input

---

Grade Breakdown:

1. **Functionality - 60%**
    a. The program runs without errors and produces the correct output for various inputs.
    b. The program meets all the specified requirements, including personalized greetings, age calculation, arithmetic challenge, and string indexing.

2. **Code Quality - 30%**
    a. Code includes meaningful comments that explain the purpose of different sections and key operations.
    b. Code is well-organized and easy to read, with proper use of whitespace and indentation.
    c. Variables are well written and are not just simple characters like ‘x’ or ‘a’
3. **Adherence to Instructions - 10%**
    a. The student followed all instructions for the assignment submission, including file naming and format.
    b. Instructions are in the syllabus