A Text Based Adventure Game

Programming Language I, CSE 115.7, Summer 2024, BCSE, Department of Electrical and Computer Engineering, North South University.
Group-01
Mashrafi Khalil Mahi 2423146642
Md. Jazib Mahamud Mahi 2422505642
Mofid Al Mustakine 2422431042
Shafiq Wayez 2422701642

1. Introduction

This project is an immersive **text-based adventure game**, meticulously designed in the C programming language, to take players on a narrative-driven journey into a mysterious, otherworldly dimension. The game intertwines the art of storytelling with logic-based challenges, engaging players in critical decision-making, problem-solving, and interactive exploration.
The idea stems from a desire to bring together **structured programming principles** and the creative process of crafting an enthralling user experience. By allowing players to shape their journey through choices, puzzles, and encounters, the project bridges technical skills and imaginative design.
This project not only serves as a demonstration of core programming skills such as loops, conditions, functions, recursion, and arrays but also explores how these fundamental tools can create meaningful, interactive, and enjoyable applications.

2. Objectives

The objectives of this project are multifaceted, encompassing both educational and creative goals:
Educational Goals:

- To apply and deepen the understanding of **C programming fundamentals** in a practical, real-world application.
- To reinforce concepts like modular programming, error handling, recursion, and input validation in a cohesive manner.
- To develop problem-solving and debugging skills through iterative coding processes and gameplay testing.

Creative Goals:

- To craft an engaging, interactive narrative where the player's choices significantly impact the storyline.
- To design logical puzzles and challenges that stimulate critical thinking and offer a rewarding experience.
- To demonstrate how text-based games can provide immersive experiences without advanced graphical components.

3. The Narrative and Gameplay Flow

3.1. Storyline Overview

The game begins with a young protagonist walking through familiar city streets, only to encounter a glowing object that transports them to an unknown dimension. Lost in this strange and surreal world, the player must navigate through unfamiliar terrains, solve riddles, and make strategic decisions to find their way home.
The storyline unfolds dynamically based on the player's decisions, offering a unique experience with every playthrough. Players are guided through various locations and scenarios, such as:

- **Misty River or Paved Road:** Initial choices determine the paths available.
- **Encounters with NPCs:** Players meet a wise scholar, a shopkeeper, and the city mayor, each presenting unique puzzles or quests.
- **Temple Trials:** A climactic sequence involving logical puzzles, a parkour challenge, and an ancient guardian.
- **The Final Test:** The player must confront an Ancient Being of Knowledge and answer its riddles to prove their wisdom.

The game concludes with a heartfelt return to the protagonist's home, where they are reunited with loved ones, symbolizing the completion of their arduous journey.

4. Development Strategy

The development of the game followed a structured and collaborative workflow:

4.1. Planning and Flowchart Creation:

As a team, we started by creating a comprehensive flowchart of the entire game, outlining its structure, key decisions, narrative branches, and functional requirements. This step ensured that everyone had a clear understanding of the game's design and flow.

4.2. Function-Based Development:

Based on the flowchart, we divided the work by functions, ensuring modularity and efficient collaboration:

- **Mithun:** Was responsible for the first three functions, which included setting up the game's

introduction and early storyline progression. Those being: void startGame(); void mistyRiver(); void guardEncounter().

- **Shafiq:** Focused on the next three functions, primarily handling decision-making mechanics and branching narratives. Those being: void gameOver(); void resetGame(); void cityEncounter().
- **Jazib:** Designed and implemented three functions centered on puzzles and interactive challenges. Those being: void mayorEncounter(); void scholarEncounter(); void shopkeeperEncounter().
- **Mashrafi:** Worked on the rest of the functions and took charge of integrating all parts, finalizing the game's structure, testing, debugging, and handling error management.

4.3. Execution and Integration:
After the functions were developed, the team collaborated to integrate the modules, test their compatibility, and ensure a seamless gameplay experience.

4.4. Final Testing and Debugging:
Rigorous testing was performed to identify and fix logical inconsistencies, edge cases, and input validation issues before finalizing the game.

This systematic workflow allowed us to divide responsibilities effectively and complete the project on time while maintaining the quality of the final product.

5. Code Structure and Key Features

5.1. Modular Design
The project adheres to a modular programming approach, where the game is divided into self-contained functions. Each function serves a specific purpose, such as handling an encounter, resetting the game, or validating input. This structure promotes code clarity, reusability, and ease of debugging.

5.2. Use of Programming Elements

i. **Loops:**
Loops are utilized to control repetitive gameplay mechanics, validate user inputs, and manage retry attempts. For example, riddles allow up to three attempts to answer correctly, with the game terminating upon failure.

ii. **Conditional Statements:**
Conditional logic determines the flow of the game based on player decisions. For instance:

- Choosing between the Misty River and the Paved Road alters the storyline.
- Encountering the Ancient Guardian leads to different outcomes depending on the player's answers.

iii. **Functions:**
Functions such as startGame(), mistyRiver(), temple(), and gameOver() compartmentalize the code, ensuring modularity and clarity. For example:

- The temple() function integrates puzzles, parkour challenges, and a seamless transition to subsequent encounters.
- The resetGame() function resets global variables, enabling players to restart the game from scratch.

iv. **Global Variables:**
Global variables are used to manage the state of the game and ensure logical consistency throughout gameplay. These variables allow the game to track player progress, visited locations, and the overall outcome of the game. Examples include:

- mistyRiverVisited**:** Tracks whether the Misty River has been visited to prevent revisiting the same location.
- returnedToFork**:** Ensures logical progression by recording whether the player has returned to a critical fork in the game.
- gameWon **:** Tracks whether the game has been successfully completed. Initially set to 0, it changes to 1 when the player completes the game successfully.

These variables streamline state management, allowing functions to reference and update critical aspects of gameplay efficiently.

v. **Input Validation:**
The game employs rigorous input validation to handle unexpected or invalid inputs gracefully. For example:

- Players are prompted to re-enter valid choices when numeric input is required.
- Non-alphabetic characters in riddle answers are detected and rejected with meaningful feedback.

vi. **Recursion:**
Recursive calls are used for restarting the game after a win or a game-over, demonstrating efficient program control while maintaining the flow of the storyline.

vii. **Randomization:**
The rand() function ensures a dynamic experience

by selecting random riddles or encounters, increasing replayability.

6. Challenges Faced During Development

The development of the text-based adventure game presented several challenges that required careful problem-solving and iteration. Below is a deeper exploration of these challenges:

6.1. Game State Management

Challenge:
The game needed to track the player's progress and ensure consistency in gameplay. This included preventing revisits to previously completed areas, tracking when the game was won, and resetting variables during a game restart.
Solution:

- Global variables such as mistyRiverVisited and returnedToFork were used to monitor specific game states. These variables ensured logical consistency and prevented players from revisiting places unnecessarily.
- The gameWon variable was introduced and initialized to 0 at the start of the game. It tracked whether the player had successfully completed the game. When the player met the victory conditions (e.g., completing the final trial), gameWon was updated to 1.
- The resetGame() function was designed to clear all progress-related variables, including gameWon, ensuring that every new playthrough began with a clean slate. This avoided issues like residual game states affecting subsequent plays.

Outcome:
This solution provided a structured way to manage the player's journey, track the game's completion state, and maintain logical consistency throughout the game.

6.2. Input Validation and Error Handling

Challenge:
Players frequently make errors, such as entering non-numeric characters for numeric prompts or providing overly long and invalid answers to riddles. Without proper handling, these could cause crashes or disrupt gameplay.
Solution:

- For numeric inputs, validation loops checked whether the input was an integer. Invalid inputs triggered error messages and allowed retries.
- For string inputs (e.g., riddle answers), a custom validation function, isValidAnswer(), ensured only alphabetic characters were accepted. It also prevented excessively long inputs, guiding players toward meaningful responses.
- Buffers were cleared of invalid characters to ensure smooth execution.

Outcome:
These mechanisms created a robust system that not only prevented crashes but also enhanced the user experience by providing clear and helpful feedback.

6.3. Logical Flow and Consistency

Challenge:
The game offered dynamic choices, allowing players to make decisions that influenced the storyline. Maintaining a logical narrative flow while accounting for all possible player actions was complex. Poorly designed logic could lead to contradictions or unintentionally abrupt endings.
Solution:

- Each function (e.g., mistyRiver(), guardEncounter()) was carefully crafted to handle specific scenarios and outcomes. Flags like returnedToFork were implemented to ensure a seamless return to the main storyline after side events.
- Revisit logic was incorporated to block redundant actions, such as revisiting a completed area like the Misty River, which triggered a specific game-over scenario.

Outcome:
Extensive testing and thoughtful narrative design ensured the storyline remained cohesive and immersive, with every decision flowing logically into the next.

6.4. Designing Challenges and Puzzles

Challenge:
Creating engaging riddles and puzzles that were neither too easy nor too difficult required a balance between complexity and accessibility. Additionally, the challenges had to fit organically within the game’s narrative.
Solution:

- Riddles were carefully selected for their logical appeal and relevance to the storyline. For example:
  - "What has to be broken before you can use it?" (Answer: Egg).
  - "I am a five-letter word. I get shorter when you add two letters to me." (Answer: Short).
- Players were given clear instructions on input format (e.g., alphabetic-only answers) and allowed up to three retry attempts for each puzzle, providing a balance between challenge and fairness.

Outcome:
The puzzles became integral parts of the storyline, adding intellectual engagement while ensuring players could progress without frustration.

6.5. Debugging Complex Scenarios

Challenge:
With branching narratives and multiple paths, debugging the game was a time-intensive process. Edge cases, such as revisiting untracked locations or exceeding retry limits, posed risks of disrupting gameplay.

Solution:

- Comprehensive testing was conducted for each scenario individually and in combination with others. This helped identify and resolve edge cases, such as:
    - Players exhausting retry attempts for puzzles or making invalid choices repeatedly.
    - Revisiting paths that weren't properly blocked.
- Default behaviors were established for unresolved scenarios, such as triggering gameOver() when players exceeded their retry limits.

Outcome:
The game achieved a stable and polished flow, even under extreme or unexpected player actions.

7. Immersive Gameplay Features

- **Interactive Choices:** Players make meaningful decisions that influence the outcome. For example, choosing to observe the Temple Warden instead of engaging in combat reveals a strategic path to victory.
- **Diverse Challenges:** From solving riddles to navigating treacherous paths, the game keeps players engaged with a variety of tasks. The parkour challenge in the temple is a highlight, offering an action-oriented diversion from puzzles.
- **Replayability:** With randomized elements and branching paths, players can replay the game to explore different outcomes and challenges.
- **Thoughtful Error Handling:** The game gently guides players through retries, avoiding frustration while maintaining stakes. Clear messages and retry limits ensure balance.

8. Reflections and Future Enhancements

Achievements:
This project successfully:

- Demonstrated the use of fundamental programming skills, including loops, conditions, and recursion.
- Provided an engaging game with a dynamic storyline and challenging gameplay.
- Showcased how basic programming constructs can create engaging applications.

Challenges Conquered:
This project required meticulous planning, logical structuring, and debugging. It highlighted the importance of breaking down problems into smaller, manageable components.

Potential Improvements:

1. **Save/Load Feature:** Implementing a feature to save player progress for longer gameplay sessions.
2. **Expanded Storyline:** new locations, characters, and challenges to deepen the narrative.
3. **Graphical Enhancements:** Incorporating basic graphics or animations for a richer experience.

9. Conclusion

This project is a testament to how creativity and technical skills can combine to produce a meaningful and engaging experience. By leveraging core programming concepts, this game bridges the gap between structured logic and imaginative storytelling. It demonstrates that even basic tools like text and code can craft worlds where players are free to explore, engage, and challenge themselves.
The journey of developing this project mirrors the game's theme: overcoming challenges, solving puzzles, and growing through experiences. It stands as a proud milestone in the journey of mastering programming and storytelling.