Assignment 4 Journal / Research / Testing

- A document in which you discuss the research you conducted to create the game, as well as:
    - your design decisions (that is, your reflections on how you arrived at the finished design);
    - the challenges you faced and how you overcame them; and
    - the testing you did and how you solved any issues found in testing.

  This document should be no less than 1000 words in length, and no more than 5000 words. You may incorporate this portion of the project into your Assignment Journal, if you prefer.

Journal

Wonderland Game Development Journal

Name: Boris Bojanov

ID: 3608550

Date Started: January 23, 2025

Date Completed: January 28, 2025

Overview of the Development Journal

This journal documents my progress in developing a text-based adventure game inspired by *Alice's Adventures in Wonderland*. The goal was to create a modular, object-oriented game system with dynamic user interaction, flexible action handling, and a scalable structure for game components like locations, characters, items, and actions. This journal reflects on the planning, design decisions, challenges, and solutions throughout the development process.

Daily Progress and Activities

Day 1 (Jan 23): Project Setup and Initial Class Design

1. **Activities**:
    - Reviewed project requirements, including mandatory classes (Game, Locations, Items, Characters, Actions, Inventory, Control).
    - Designed the initial class structure with headers and basic functionality.
    - Created placeholders for `Actions`, `Control`, and `Game` classes to establish the basic flow.
2. **Challenges**:
    - Deciding how to manage interactions between `Control`, `Actions`, and `Game` while maintaining clear separation of concerns.
    - Mapping how actions would affect other components like inventory and locations.
3. **Solutions**:
    - Planned to use a centralized `Control` class to parse user input and dispatch it to `Actions` for processing.
    - Used placeholder methods for inventory management and location updates.

Day 2 (Jan 24): Input Parsing and Action Execution

1. **Activities**:
    - Implemented `Control::processInput` to parse user input into an action and an object.
    - Integrated input parsing with `Actions::executeAction` to handle commands like "take flashlight" or "go north."

- Tested basic input-output flow to ensure the parsed commands were correctly routed.

2. **Challenges**:
   - Edge cases where user input included extra spaces or invalid commands.
   - Managing multi-word objects (e.g., "golden key") while preserving command clarity.
3. **Solutions**:
   - Added input trimming and robust parsing logic to split the input into an action and object dynamically.
   - Used a helper function to handle multi-word objects gracefully.

Day 3 (Jan 25): Dynamic Action Handling with Functors

1. **Activities**:
   - Refactored the `Actions` class to use a map of callbacks for dynamic action registration.
   - Registered core actions (e.g., "take," "drop," "inventory") with corresponding functors to handle their logic.
   - Tested the dynamic dispatcher by adding and executing custom actions without modifying the core code.
2. **Challenges**:
   - Understanding how to pass object-specific context into the callbacks without overcomplicating the design.
   - Debugging issues where unregistered actions caused unexpected behavior.
3. **Solutions**:
   - Used `std::function` to allow flexible callbacks with captured state for dynamic behavior.
   - Improved error handling by validating actions before execution and providing descriptive error messages.

Day 4 (Jan 26): Expanding Game Components (Locations and Inventory)

1. **Activities**:
   - Designed and implemented the `Locations` class to store room descriptions and connections.
   - Extended the `Actions` class to include inventory management logic (`addToInventory`, `removeFromInventory`).
   - Began integrating location transitions (e.g., "go north") into the action system.
2. **Challenges**:
   - Deciding how to represent location connections dynamically and efficiently.
   - Handling invalid transitions (e.g., moving to non-existent or blocked locations).
3. **Solutions**:
   - Used a map structure to store location connections (e.g., {"north": "Rabbit Hole"}).
   - Added validation logic in `Actions::updateLocation` to check if a transition was valid.

Day 5 (Jan 27): Debugging and Refinements

1. **Activities**:
   - Addressed bugs in input parsing and action execution, including mismatched actions and objects.
   - Refactored `Control::handleUserInput` to improve readability and maintainability.
   - Added detailed debug output to track the flow of actions and object interactions.

2. **Challenges**:
    - Incorrect parsing of user input caused invalid action-object combinations.
    - Overhead from repetitive debug output made tracking progress difficult.
3. **Solutions**:
    - Centralized debugging with clear, consistent messages for input, parsing, and action execution.
    - Updated parsing logic to handle edge cases (e.g., extra spaces, invalid commands).

Day 6 (Jan 28): Final Integration and Testing

1. **Activities**:
    - Integrated all components (`Game`, `Control`, `Actions`, `Locations`) into a cohesive system.
    - Conducted extensive testing for common actions ("take," "drop," "go north") and validated multi-word object handling.
    - Implemented dynamic action registration to allow future extensibility.
2. **Challenges**:
    - Ensuring seamless interaction between components while maintaining object-oriented principles.
    - Handling corner cases like empty input, invalid locations, and unregistered actions.
3. **Solutions**:
    - Added robust error handling for invalid inputs and state transitions.
    - Documented the final structure to facilitate future enhancements and debugging.

Final Reflections

This project was an excellent exercise in **object-oriented design** and **dynamic behavior implementation**. The use of functors and callback functions significantly enhanced the flexibility of the `Actions` class, enabling a scalable and modular architecture. Key takeaways include:

- **Dynamic Action Dispatch**: Using a map of callbacks streamlined the process of adding and managing actions.
- **Separation of Concerns**: Clearly defined roles for `Control`, `Actions`, and `Game` ensured maintainable and testable code.
- **Robust Input Parsing**: Careful handling of edge cases improved the user experience and minimized errors.

Future Improvements:

1. Expand the `Locations` system to include richer descriptions and interactive elements.
2. Add support for saving and loading game states.
3. Implement advanced action logic (e.g., interacting with characters or combining items).

This journal has been instrumental in documenting the development process, highlighting challenges, and reflecting on solutions, ultimately leading to a deeper understanding of object-oriented principles and dynamic programming techniques.

Sources and References

1. C++ Reference – for `std::function`, string handling, and STL usage.
2. DevDocs C++ https://devdocs.io/cpp/
3. W3School https://www.w3schools.com/cpp/

Research

Burton, T. (2010). Alice in Wonderland. Walt Disney Studios Motion Pictures.

Crowther, W., & Woods, D. (1977). Colossal Cave Adventure [Video game]. Self-published. https://en.wikipedia.org/wiki/Colossal_Cave_Adventure

Carroll, Lewis. (2006). Alice in Wonderland. Urbana, Illinois: Project Gutenberg. https://www.gutenberg.org/

The research for this project was conducted to improve my understanding of how the game should feel and what kind of information should be relayed to the player.

Several key insights emerged from this research process:

1. **Sparse Guidance and Player Agency**

   Inspired by *Colossal Cave Adventure,* this design choice encourages exploration and problem-solving, for example, while the game informs players that their objective is to locate the golden key and reach Room 16, it leaves the path and strategy up to the player.

2. **Atmosphere and Environment**

   Drawing from Burton's film and Carroll's original text, the game emphasizes well described locations and a sense of whimsies. Each's room text is designed to reflect Wonderland's nature.

3. **Puzzle Design and Challenges**

   The objective of finding the golden key and navigating to Room 16 directly reflects the Hall of Doors sequence in *Alice's Adventures in Wonderland.* Puzzle is designed to mimic the nonsensical yet logical progression of Wonderland.

Testing

Testing was an integral part of the game development process, ensuring that all components worked together seamlessly and provided a positive player experience. The testing process included functionality validation, play-testing with users, and error-handling evaluation.

---

1. Class Validation and Command Testing

- Each class (e.g., `Game`, `Control`, `Actions`, `Locations`) was tested individually to confirm that its methods behaved as expected. For example:

  - `Actions` was validated by entering a variety of commands, such as "take flashlight," "go north," and "drop key," to ensure they executed the corresponding game logic correctly.
  - `Control` was tested to ensure that user inputs were parsed accurately, and actions were dispatched to the correct components.
  - `Locations` was tested to verify valid and invalid room transitions, ensuring the game's navigation logic matched the intended map structure.
- Edge cases were also tested, such as:
  - Entering commands with extra spaces or mixed capitalization (e.g., " Go north " or "DROP Key").
  - Providing incomplete commands (e.g., "take" without specifying an object).

---

2. Playtesting with External Users

- To evaluate how players unfamiliar with the project would interact with the game, two friends were invited to playtest it.
- Their feedback highlighted usability issues, including:
  - Confusion about available commands and valid objects.
  - Difficulties navigating the map without clearer hints or instructions.
- Based on their feedback, the following adjustments were made:
  - Added contextual hints (e.g., listing available exits in each room).
  - Improved error messages to provide more guidance when invalid commands were entered.

---

3. Intentional Input Limitations

- Input validation was implemented to restrict user inputs to predefined commands and objects. For example:
  - Commands like "take" and "drop" only work with valid items in the game world.

    - Directional commands like "go north" require the specified direction to be valid based on the player’s current location.
- These limitations were intentionally designed to simplify input parsing and focus the player’s interactions on the core mechanics.

---

4. Error Handling and Known Issues

- The game was tested for various error conditions, including invalid commands, attempts to interact with non-existent objects, and navigation to invalid locations. While basic error handling is implemented (e.g., displaying "Unknown command" or "You can’t go that way"), there are areas for improvement:
    - **Error Feedback**: Some error messages could be more specific and helpful.
    - **Graceful Recovery**: Currently, the game terminates if a critical error occurs, such as failure to load a required file. Adding fallback mechanisms or prompts for retrying could improve robustness.

Reflections on Testing

Testing revealed both strengths and opportunities for improvement:

- The game’s modular design made it easier to isolate and test individual components, ensuring that bugs were addressed systematically.
- Playtesting provided valuable insights into how real users interact with the game, emphasizing the importance of clear feedback and guidance.
- While the input validation successfully limited invalid commands, expanding the parser to handle a wider variety of input styles (e.g., synonyms or more flexible phrasing) could enhance accessibility.

Next Steps:

- Improve error handling by providing more detailed feedback and enabling recovery from critical failures.
- Expand the command parser to handle synonyms (e.g., "grab" for "take") and more complex input.

- Conduct additional playtests to refine the user experience and identify further enhancements.