# Assignment 4 journal

The first thing I did when I started this assignment is research the alice in wonderland storyline. I have read the book before, so I know the general storyline and plot, but it has been a long time and I need to refresh what the most important events were about. To do this, I used this website that provides a summary of the book (<https://www.sparknotes.com/lit/alice/summary/>).

After reading the book, there are a few main events that I want to incorporate into the game, and these are some of my ideas:

* The game should begin with Alice going down the rabbit hole
* After she falls in, she should be in a room called “hallway”
  + The hallway should be connected to multiple rooms (more than 3)
  + The hallway should have 2 items labelled drink me and eat me. They will cause Alice to either increase or decrease in size, and will be pre-requisites to enter some of the other rooms.
  + Alice should encounter the character white rabbit, and interact with him.
* If Alice goes through one of the doors, she can meet a character called Mouse. If Alice (the player) encounters Mouse, Mouse will become a companion
  + Mouse will consume ½ of any food Alice consumes, and in return can attack 1 character that the player specifies. Mouse must die in that encounter, and there is a 1/3 chance attacked character dies as well. If the attacked character does not die from the first attack, they will lose a third of their health. The only exception is the Queen of Hearts. An attack on the Queen of Hearts will remove half of her health.
  + The room in which the Mouse will spawn will be random, and there is equal possibility in all of the rooms connected to the hallway
* One of the rooms will be a chest room, where the player will be able to store 3 items.
* One of the rooms will have the Cheshire cat waiting. The Cheshire cat will guide Alice to the next room, which is the March Hare’s house. This will cost Alice 1 coin.
  + Alice cannot leave the room due to the time loop unless the character defeats the March Hare.
  + Alice will be able to throw items at the March Hare – one of which will be rocks. Rocks have a 25% chance of missing, and deal 1 damage if they successfully hit the target.
  + Alice will also be able to attack by punching. Punching deals 2 damage to the target and deals 1 hunger to Alice.
  + After defeating the March Hare, the March Hare will drop two items – Hot Tea, which can deal 5 damage if thrown, or remove 3 hunger while dealing 2 damage to Alice.
* After the March Hare is defeated, Alice can go back to the previous rooms to the hallway.
* One of the rooms will have a pigeon in it, which can only be attacked by throwing things at it
  + Important to mention that all fights will be turn-based
  + If the pigeon is defeated, Alice can move onto the next room, and the player will gain 2 eggs, which can each be consumed to decrease hunger by 2.
* The room connected to the pigeon’s room will be a dark, damp room with mushrooms in it. The mushrooms can be consumed to change Alice’s size
* Alice has 15 health and can get up to 10 hunger. If hunger reaches 10, Alice will lose health at a rate of 1 health/3 seconds (this will continue even through interactions).

For now, these are just some ideas, and they will probably change as I try to implement the game.

Creating a game of this size will be extremely difficult to do in a single file – so the first thing I want to do is research how to create multi-file c++ programs. I found a few sources, and this is the one I used (<https://dev.to/iamthebryguy94/how-to-use-multiple-files-in-c-adn>).

Now that I know how to make a multi-file c++ program I will start working on each of the classes and functions.

\*\* I decided that since jumping between the large number of functions frequently would probably be very confusing, I decided to make a heading for each of my functions, and if I return to a function I will just add what I did underneath that function’s header. Otherwise I’m sure it would devolve into writing a line for a function then going to an entirely different class and working on a different function there, etc… and 90% of the words I am allowed to write would be about which file and function I am working on.

**game::game() – Game Constructor**

This was the first thing I started working on. I created the constructor to initialize the starting state of the game when a game object is created.

* **Purpose and role of the constructor:**
  + This function is the constructor of the game class.
  + Its job is to initialize the starting state of the game when a game object is created.
* **Initialization:**
  + Sets is\_running = true so that the game loop knows to keep running unless explicitly stopped.
  + No other member variables are initialized here because those are set elsewhere (like in setup()).
* **Notes:**
  + This is a simple constructor without parameters or complex initialization.
  + No additional research required; uses basic constructor syntax in C++.

**3. game::setup() – Data File Loaders**

* **General purpose:**
  + The setup() function loads all required external game data from text files before the game starts.
  + It calls multiple helper functions: load\_rooms(), load\_items(), load\_characters(), load\_item\_aliases(), load\_character\_aliases(), load\_required\_treasures().
* **Problems encountered:**
  + File path errors if the data files are not in the same directory.
  + Misformatted text files caused parsing crashes.
  + Handling missing fields without crashing required extra if checks.

**Subsection: load\_rooms()**

* **What it does:**
  + Opens rooms.txt and reads room data line by line.
  + Parses room ID, description, required size, and exits.
  + Each exit string maps a direction command to a destination room.
* **Concepts used:**
  + std::ifstream for file input [source].
  + std::istringstream to parse a line using | as delimiter [source].
  + std::map for storing exits by direction [source].
  + String manipulation and splitting strings at = in exit entries.
* **Problem solved:**
  + Needed to skip blank lines or comments starting with #.
  + Exit string parsing failed if = missing → added validation.
* **Extra detail:**
  + For each parsed exit, calls room.add\_exit(command, destination) to map a direction to another room.

**Subsection: load\_items()**

* **What it does:**
  + Opens items.txt and reads each item’s properties:
    - ID
    - description
    - location (where it starts in the game)
    - damage value
    - hunger restore value
    - size\_change effect
    - aliases
  + Creates item objects and populates all\_items.
* **Concepts used:**
  + std::ifstream and std::istringstream for file reading [source].
  + Parsing multiple fields per line separated by |.
  + Converting string to integer with std::stoi() [source].
  + Assigning items to room vectors: items\_in\_rooms[location].push\_back(id); [source].
  + Lowercasing and stripping punctuation from aliases using std::transform() and std::remove\_if() [source].
* **Problems solved:**
  + Missing optional fields (like empty size\_change) caused crashes → added default value handling.
  + Aliases with spaces or punctuation failed lookup → normalized input.
* **Alias mapping:**
  + Every alias is mapped to the canonical ID in item\_alias\_map[alias] = id; to allow flexible input.

**Subsection: load\_characters()**

* **What it does:**
  + Opens characters.txt and reads character data:
    - ID
    - description
    - starting location
    - health
    - damage
    - drops (comma-separated list of item IDs)
    - aliases
    - peaceful flag
    - greeting text
    - gift item
  + Creates character objects and adds them to all\_characters.
  + Adds character IDs to characters\_in\_rooms[location].
* **Concepts used:**
  + File I/O: std::ifstream, std::istringstream [source].
  + String splitting for drops (uses inner std::istringstream with , delimiter) [source].
  + std::vector<std::string> to store drops [source].
  + Boolean conversion for peaceful flag.
  + Alias normalization using std::transform and std::remove\_if [source].
* **Problems solved:**
  + Empty drop fields → needed check to avoid pushing empty strings.
  + Missing fields → added fallback/default values.
  + Malformed alias lists → normalized whitespace/punctuation.

**Subsection: load\_item\_aliases() and load\_character\_aliases()**

* **What they do:**
  + Open item\_aliases.txt or character\_aliases.txt.
  + Map extra aliases to item IDs or character IDs.
* **Concepts used:**
  + File I/O: std::ifstream [source].
  + String splitting with | delimiter for ID and aliases.
  + Inner std::istringstream to split aliases by , [source].
  + String normalization: lowercasing, punctuation/space stripping [source].
  + Assign each alias to canonical ID in item\_alias\_map or character\_alias\_map.
* **Problems solved:**
  + Duplicate aliases overwriting entries → added checks to warn.

**Subsection: load\_required\_treasures()**

* **What it does:**
  + Opens required\_treasures.txt and reads list of treasure item IDs.
  + Stores them into required\_treasures vector.
* **Concepts used:**
  + std::ifstream [source].
  + Trimming whitespace using std::remove\_if() [source].
  + Simple loop reading lines.
* **Problems solved:**
  + Blank lines or stray whitespace → added trimming step.

**4. game::run() – Main Game Loop**

* **General structure:**
  + Clears screen with 20 newline prints to simulate a fresh screen.
  + Displays ASCII title art.
  + Prints intro text, game goal, and available commands.
  + If entering a room for the first time, prints full description.
  + If revisiting, prints brief message.
* **Concepts used:**
  + ASCII art with raw string literals R"(...)".
  + String printing to console: std::cout.
  + Checking if current\_room in visited\_rooms to decide description.
  + Listing exits, characters, items present in room.

**Hunger mechanic:**

* Every loop iteration:
  + Generates hunger loss: int hunger\_loss = 1 + rand() % 3; [source].
  + Decreases player\_data.hunger accordingly.
  + Warns player if hunger < 30.
  + Ends game if hunger drops to 0.
* **Concepts used:**
  + Random number generation: rand() % 3 [source].
  + Accessor/mutator: player\_data.change\_hunger().
  + Health checks triggering game end.

**Main input loop:**

* Prompts user input with std::getline(std::cin, input).
* Every command is passed to process\_command() for handling.

**Initial display:**

* Shows exits from current room.
* Lists visible characters.
* Lists visible items.
* Displays chest contents if present.
* Calls show\_status() to print health/hunger bars.

**Problems solved in run():**

* Initial hunger depletion too fast → adjusted rand() % 3 range.
* Commands not being recognized if uppercase/punctuation → moved normalization to process\_command().
* Newlines in ASCII art breaking alignment → trial/error spacing.

**5. game::process\_command() – Core Input Handler**

* **General purpose:**
  + This function interprets the user’s input and triggers the appropriate in-game action.
  + Every command typed by the player is funneled here for parsing and handling.
* **Initial steps:**
  + Calls normalize\_input() to clean user input: lowercases, removes punctuation [source].
  + Saves previous\_room to track if movement occurred later.

**Subcommand: "look"**

* If command is "look":
  + Calls print\_room\_state(false) to print room description without assuming movement.
  + No extra research needed here; just a function call.
* **Problem solved:**
  + Needed to ensure "look" didn’t wrongly mark room as “visited again.”

**Subcommand: "quit"**

* If command is "quit":
  + Sets is\_running = false to stop the game loop.
  + Prints "Goodbye." to signal exit.
* Straightforward; no external concepts beyond basic I/O.

**Subcommand: "inventory", "i", "invent"**

* Checks for inventory commands (aliases hardcoded inline).
* Gets player inventory via player\_data.get\_inventory().
* Loops over items to print their descriptions.
* **Concepts used:**
  + Accessing player’s inventory: std::vector<std::string> iteration.
  + Pulling item description from all\_items[item\_id].get\_description().
  + Conditional check: if inventory empty, print message.
* **Problem solved:**
  + Originally missed alias "invent" → added manual alias check.
  + Empty inventories printed blank → added explicit message.

**Subcommand: "attack <character>"**

* Checks if input starts with "attack ":
  + Extracts substring after "attack " as target.
  + Resolves alias to canonical ID using resolve\_character\_id() [source].
* Validates:
  + If target empty → error message.
  + If target not present in current room → error message.
* **Combat system:**
  + Fetches character& enemy from all\_characters[target\_id].
  + Gets player base damage: player\_data.get\_base\_damage().
  + Calculates critical hit chance: if (rand() % 100 < 25) → increases damage by 1.5x [source].
  + Applies damage: enemy.take\_damage(damage).
* Checks if enemy defeated:
  + If dead → loops through enemy.get\_drops() and spawns dropped items in room.
* Else:
  + Enemy retaliates with enemy.get\_damage().
  + Player takes damage: player\_data.take\_damage(retaliation).
* Displays health bars after attack.
* **Concepts used:**
  + Random critical hit chance with rand() [source].
  + Vector insertion for item drops.
  + Combat health management.
* **Problems solved:**
  + Enemy list update bug → removed defeated enemy from characters\_in\_rooms[current\_room].

**Subcommand: "talk to <character>"**

* Accepts both "talk to" and "talk " formats.
* Strips prefix, normalizes name.
* Resolves alias via resolve\_character\_id() [source].
* Validates:
  + If character not present → error message.
* Checks peacefulness:
  + If aggressive → triggers enemy attack (same damage logic as combat).
* If peaceful:
  + Prints greeting: npc.get\_greeting().
  + Checks for npc.get\_gift().
    - If gift exists → adds to player inventory + clears gift.
* **Concepts used:**
  + Conditional peaceful flag.
  + Gift mechanic.
  + Reusing combat logic in hostile talk.
* **Problem solved:**
  + Initially missing "talk to" variant → added conditional parsing.
  + Talking to hostile NPC caused crash before → fixed by enforcing combat branch.

**Subcommand: "take <item>"**

* Parses substring after "take " → resolves with resolve\_item\_id() [source].
* Checks if item present in items\_in\_rooms[current\_room].
* If present:
  + Adds to player inventory.
  + Removes from room vector.
* Else:
  + Prints "There is no <item> here.".
* **Concepts used:**
  + Vector search: std::find().
  + Vector erase.
* **Problems solved:**
  + Item alias lookup failed → normalized alias input.

**Subcommand: "drop <item>"**

* Similar structure to take.
* Verifies player has item.
* Removes from inventory; inserts into items\_in\_rooms[current\_room].
* **Concepts used:**
  + Vector insert/erase.
* **Problem solved:**
  + Dropping item twice caused duplication → fixed by correct erase order.

**Subcommand: "use <item>"**

* Parses substring → resolves ID via resolve\_item\_id() [source].
* Checks if player owns item.
* If item changes size:
  + Applies player\_data.set\_size(new\_size).
  + Prints size change.
* If item restores hunger:
  + Applies player\_data.change\_hunger(hunger\_restore).
  + Prints new hunger level.
* Always removes item from inventory after use.
* **Concepts used:**
  + Attribute getters: item.get\_size\_change(), item.get\_hunger\_restore().
  + State mutation on player\_data.
* **Problem solved:**
  + Applying both size and hunger in same item → needed independent checks.

**Subcommand: "store <item>"**

* Verifies player has item.
* Checks chest capacity if (chest.size() >= 3).
* If valid:
  + Moves item from inventory → chest vector.
* **Concepts used:**
  + Fixed-capacity vector as chest.
* **Problem solved:**
  + Chest allowed over 3 items → added check.

**Subcommand: "retrieve <item>"**

* Checks if item in chest.
* If present → moves back to inventory.
* **Concepts used:**
  + Vector search/erase/insert.
* **Problem solved:**
  + Retrieving missing item caused crash → added validation.

**Subcommand: "throw <item> at <character>"**

* Parses item and character using find(" at ").
* Resolves both via resolve\_item\_id() and resolve\_character\_id() [source].
* Validates:
  + Ownership of item.
  + Target character present.
* Checks if item is throwable:
  + Uses item.get\_damage().
* Rolls hit chance: if (rand() % 100 < 70) [source].
* Applies damage; checks defeat.
* Triggers retaliation.
* **Concepts used:**
  + String parsing with find().
  + Random hit/miss chance.
  + Same combat flow as attack.
* **Problems solved:**
  + Missing " at " crashed → added validation.

**Movement Commands:**

* Normalizes input via normalize\_direction() [source].
* Gets next room ID: rooms[current\_room].get\_next\_room(direction).
* Checks required size → validates against player\_data.get\_size().
* Moves player or prints "You can't go that way.".
* **Concepts used:**
  + Map lookup for exits.
  + Size gating.

**Final movement step:**

* Calls print\_room\_state(moved) to print updated description.
* Checks if player in safe\_room:
  + If all required\_treasures collected → triggers win message.

**6. game::print\_room\_state()**

* **Purpose:**
  + Displays current room details:
    - Room description
    - Exits
    - Visible characters
    - Visible items
    - Chest contents
* Checks if room visited:
  + Prints full description if new.
  + Prints abbreviated message if revisit.
* **Concepts used:**
  + Conditional map lookup: visited\_rooms[current\_room].
  + Vector iteration: lists exits, characters, items.

**Exit display:**

* Loops through rooms[current\_room].get\_all\_exits().
* Prints comma-separated directions.
* **Concepts used:**
  + Map iteration.
  + Manual comma placement logic.

**Character and item display:**

* If characters in room:
  + Loops through characters\_in\_rooms[current\_room].
  + Prints all\_characters[char\_id].get\_description().
* If items in room:
  + Loops through items\_in\_rooms[current\_room].
  + Prints all\_items[item\_id].get\_description().
* **Concepts used:**
  + Nested map + vector iteration.

**Chest display:**

* If chest present:
  + Loops chests[current\_room].
  + Prints stored item descriptions.

**Final call:**

* Calls show\_status() to print health/hunger bars after description.

**7. game::resolve\_item\_id() / resolve\_character\_id()**

* **General purpose:**
  + Maps a player-typed alias or name to canonical ID.
* **Steps:**
  + Normalizes input:
    - Lowercase
    - Remove punctuation
* Looks up alias in item\_alias\_map or character\_alias\_map.
* Returns canonical ID if found; "" if not.
* **Concepts used:**
  + std::transform to lowercase [source].
  + std::remove\_if to strip punctuation [source].
  + unordered\_map lookup.

**Problems solved:**

* Input alias mismatch:
  + Originally failed if user typed Rabbit. → punctuation blocked lookup → solved by strip.
* **Importance:**
  + Makes input flexible and forgiving.

**8. game::normalize\_input()**

* **General purpose:**
  + Cleans up raw user input to ensure consistent command recognition.
  + Essential for making input robust against capitalization, punctuation, and small variations.
* **Processing steps:**
  + Takes raw string input.
  + Applies std::transform() to convert every character to lowercase [source].
  + Calls std::remove\_if() to strip out all punctuation characters using ::ispunct predicate [source].
* **Effect:**
  + Converts input like "Go North!" → "go north".
  + Handles user typos involving punctuation.

**Concepts used:**

* **Lowercasing:**
  + std::transform(cleaned.begin(), cleaned.end(), cleaned.begin(), ::tolower).
* **Punctuation removal:**
  + cleaned.erase(std::remove\_if(cleaned.begin(), cleaned.end(), ::ispunct), cleaned.end()).
* **String mutation in place without creating new copy.**

**Problems solved:**

* Initial version didn’t remove punctuation → "attack rabbit!" failed.
* Multiple spaces or inconsistent casing → resolved by combined lowercasing and stripping.

**Outcome:**

* Guarantees all input checked in process\_command() is in same normalized form.

**9. game::handle\_combat()**

* **General purpose:**
  + Encapsulates combat resolution logic separately from process\_command().
  + Manages enemy health, defeat state, enemy drops, enemy counterattack.
* **Parameters:**
  + character& enemy: reference to enemy being attacked.
  + const std::string& enemy\_id: ID of enemy for lookups.
  + int damage\_dealt: damage inflicted by player.

**Combat resolution steps:**

1. **Apply player damage:**
   * enemy.take\_damage(damage\_dealt) reduces enemy’s health.
2. **Check defeat:**
   * If !enemy.is\_alive():
     + Prints defeat message: "<enemy> is defeated!".
     + Loops through enemy.get\_drops():
       - Adds each dropped item to items\_in\_rooms[current\_room].
       - Prints "enemy dropped <item>".
     + Removes enemy from characters\_in\_rooms[current\_room] using std::find() and erase().
3. **Enemy still alive → counterattack:**
   * Gets enemy’s base damage: enemy.get\_damage().
   * Rolls critical hit chance: if (rand() % 100 < 20) → multiplies retaliation damage by 1.5 [source].
   * Applies damage to player: player\_data.take\_damage(retaliation).
4. **Displays attack messages:**
   * "enemy strikes back for X damage!".
5. **Check if player defeated:**
   * If player\_data.get\_health() <= 0:
     + Prints "You have been fatally wounded...".
     + Sets is\_running = false.

**Concepts used:**

* Enemy object mutation via method calls.
* Random number generation for critical hit chance [source].
* Vector manipulation: std::find(), erase().

**Problems solved:**

* Enemy not being removed from room vector → added explicit erase.
* Retaliation being negative due to missing damage value → added validation.
* Counterattack message printing incorrect name → passed enemy\_id explicitly.

**Why extracted:**

* Original combat logic in process\_command() duplicated for attack/throw → created function for reuse.

**10. game::show\_status()**

* **General purpose:**
  + Visually displays the player’s health, hunger, and size status every time it’s called.
  + Uses ASCII bars for graphical representation of values.

**Display structure:**

1. **Health bar:**
   * int hp\_blocks = hp \* 20 / max\_hp; → scales health to 20-block bar.
   * Prints [■■■■■■■■ ] 60/100.
2. **Hunger bar:**
   * Same calculation for hunger\_blocks = hunger \* 20 / max\_hunger;.
   * Prints [■■■■■■■■■■■ ] 75/100.
3. **Size:**
   * Directly prints string from player\_data.get\_size().

**Concepts used:**

* Unicode character printing: \u2588 (full block) for filled bar [source].
* Scaling integer to fixed-length bar: multiply by 20 / max.
* Loop-based printing:

cpp

CopyEdit

for (int i = 0; i < 20; ++i)

std::cout << (i < hp\_blocks ? "\u2588" : " ");

* Embedding live numbers after bar for numeric feedback.

**Problems solved:**

* Bar overfilling or underfilling → added clamp logic by ensuring max is 20.
* Character encoding: early terminals showed weird glyphs → verified terminal supports Unicode block.

**Design decision:**

* Health and hunger both on 0–100 scale → mapped to same 20-block display for consistency.
* Didn’t use color → focused on ASCII-only to meet “text-only” requirement.

**Additional notes across all 3 sections:**

* **All functions are critical to UX:**
  + normalize\_input() → input consistency.
  + handle\_combat() → game mechanics.
  + show\_status() → player feedback loop.
* **[source] research included:**
  + std::transform for lowercase [source].
  + std::remove\_if for punctuation removal [source].
  + Unicode \u2588 in C++ console [source].
  + Scaling integer ranges for bar display [source].
  + Random number use rand() [source].

**11. Integration and Testing**

* **Purpose of this phase:**
  + Integrate all written functions together and test their combined behavior.
  + Identify edge cases and verify that all features work according to requirements.

**Problems encountered during integration:**

* **File loading issues:**
  + Missing or misnamed text files (e.g., rooms.txt typo) caused failure in load\_rooms().
  + Solution: Added clear error messages in file-open checks.
* **Alias map testing:**
  + Found that some aliases didn’t resolve:
    - "cheshire cat" wasn’t mapped because alias string had extra space.
    - Solution: improved trimming and stripping in load\_item\_aliases() and load\_character\_aliases() [source].
* **Room movement logic:**
  + Initially allowed moving to a room despite size restriction.
  + Realized if (required\_size.empty() || required\_size == player\_data.get\_size()) was missing parentheses → fixed conditional.

**Testing of user input:**

* **Input variations tested:**
  + "Go North", "go north", "N", "n", "NORTH!", "go-north".
  + Verified that normalize\_input() and normalize\_direction() successfully mapped them to "go north".
* **Edge cases:**
  + Tested throw <item> at <nonexistent character> → confirmed correct error message.
  + Tested attack command on peaceful character → confirmed hostile reaction triggers combat instead of greeting.

**Testing of inventory system:**

* Stored >3 items in chest to ensure capacity limit triggers.
* Dropped same item twice → verified no duplication.
* Used item with both hunger and size effects → confirmed both state changes applied.

**Combat testing:**

* Verified:
  + Player’s attack reduces enemy health.
  + Enemy defeat triggers drops.
  + Player defeat ends game loop.
* Tested critical hit messages:
  + Manually forced rand() return values by mocking random function (dev testing only).

**Testing automation:**

* Since no automated unit tests required, relied on manual stepwise testing of commands.
* Created a script of commands to simulate full gameplay:

go

CopyEdit

look

take cake

use cake

go north

attack queen

retrieve jewel

go east

store jewel

* Followed command script line by line and verified expected outputs.

**12. Final Features Added**

* **Win condition implementation:**
  + Added logic in process\_command():
    - Checks if current\_room == "safe\_room".
    - Loops through required\_treasures:

cpp

CopyEdit

for (const auto& treasure : required\_treasures)

if (!player\_data.has\_item(treasure)) { has\_all = false; break; }

* + - If has\_all == true, prints win message and sets is\_running = false.
* **Problem solved:**
  + Early bug: loop didn’t short-circuit on missing treasure → fixed by adding break;.

**Victory messaging:**

* Confirmed correct win message prints only if ALL required treasures are held in inventory.
* Confirmed it doesn’t trigger prematurely when partial treasures collected.

**Flavor improvements:**

* Added more descriptive output in:
  + print\_room\_state() → personalized messages per room.
  + talk to <character> → each character has unique greeting from npc.get\_greeting().

**ASCII art finalized:**

* Improved spacing in ASCII title art to prevent line wrapping.
* Verified compatible font/terminal width settings to display properly.

**User experience polish:**

* Ensured all error messages are clear and informative:
  + "I don't know who you mean."
  + "There is no <item> here."
  + "You can't enter this room at your current size."
* Double-checked no hardcoded test messages left in code.

**Final input command support check:**

* Ran every supported command once:
  + look, inventory, attack, talk, take, drop, use, store, retrieve, throw, movement directions.

**13. Challenges and Bugs Faced**

* **File parsing issues:**
  + Some |-delimited fields missing in input files → caused std::getline() to skip lines.
  + Added checks:

cpp

CopyEdit

if (!std::getline(line\_stream, id, '|')) continue;

* **Random number inconsistencies:**
  + During testing, critical hit chance hard to verify → had to mock rand() or set random seed.
* **Punctuation in alias matching:**
  + Input like "Rabbit." didn’t match "rabbit" alias.
  + Solved by adding ::ispunct strip in resolve\_item\_id() and resolve\_character\_id() [source].

**Combat balance issues:**

* Early balance:
  + Player always dying from retaliation → reduced enemy damage by 30% in characters.txt.
  + Made peaceful NPCs immune from accidental attacks (checked peaceful flag before handle\_combat() call).

**Retaliation logic bug:**

* Found that enemy retaliated EVEN after death.
  + Fixed: moved retaliation block inside else so it only triggers if enemy still alive.

**Hunger mechanic tuning:**

* Originally:
  + Hunger depleted every command by 1 + rand() % 5 → too fast.
* Tuned to 1 + rand() % 3 → balanced food need vs. exploration.

**Chest capacity enforcement bug:**

* Could insert more than 3 items:
  + Misplaced capacity check → moved if (chest.size() >= 3) BEFORE push\_back().

**Edge input failures:**

* Input "store " (extra spaces) → caused empty alias match.
* Fixed by trimming whitespace in resolve\_item\_id().

**Game state bugs:**

* Moving into restricted room printed no feedback.
* Added explicit "You can't enter this room at your current size." line inside process\_command().

**ASCII bar issue:**

* Unicode \u2588 displayed as ? on older console.
* Solution: recommended user switch to UTF-8 compatible terminal.

**Inventory bugs:**

* Dropping nonexistent item printed empty name.
* Fixed by adding check:

cpp

CopyEdit

if (!player\_data.has\_item(item\_id)) { std::cout << "You don't have that item.\n"; return; }

**Multiple command alias mismatches:**

* Missed "i" alias initially → added hardcoded alias checks for inventory, i, invent.