**Alice in Wonderland Text-Based Adventure Game –Design Document**

**Game Concept and Overview**

This game is a text-based adventure implemented in C++ using object-oriented programming. It divides functionality across classes representing the player, items, characters, rooms, and game control logic. The player navigates rooms, interacts with characters and items, manages health and hunger, and issues commands to progress. Inputs are normalized for flexible command handling. Outputs are textual descriptions printed to the console.

Standard C++ libraries used: <iostream>, <fstream>, <sstream>, <string>, <vector>, <map>, <unordered\_map>, <algorithm>, <cstdlib>, <thread>, <chrono>. fstream handles file I/O; sstream parses strings; iostream handles input/output streams; unordered\_map resolves aliases; thread and chrono enable timed delays.

**Game Goals**

* Collect all required treasures.
* Return treasures to the designated safe room.
* Prevent death by maintaining positive health and hunger values.
* Access size-restricted rooms by using size-altering items.

**Core Gameplay Systems**

* **Rooms:** Each room is identified by an id and contains a description, optional size\_required to enter, and a map of exits linking to other rooms. Methods allow querying room details and managing exits.
* **Items:** Items have an id, description, damage value, hunger\_restore value, and size\_change. Items are initialized from a text file. Methods return these properties.
* **Characters:** Characters have an id, description, health, damage, a list of drop\_items, a peaceful flag, a greeting, and an optional gift\_item. Characters are initialized from a text file. Methods support combat, status queries, and interactions.
* **Player:** Tracks health, hunger, size, base\_damage, and an inventory of item IDs. Methods manage stats and inventory.
* **Inventory/Chests:** Player inventory holds any number of items; room chests store up to 3 items per room.
* **Input Normalization:** Cleans input text by lowercasing, stripping punctuation, and mapping aliases.
* **Command Processing:** Interprets normalized input and calls the corresponding game action.

**4. Class Breakdown with Detailed Function Explanations**

**4.1 Game**

**File:** game.cpp, game.h

**Functions:**

* setup(): Opens each required text file. Calls load\_rooms(), load\_items(), load\_characters(), load\_item\_aliases(), load\_character\_aliases(), load\_required\_treasures(). Initializes the core data structures: map containers store rooms, items, characters; unordered\_map stores alias mappings.
* run(): Prints ASCII title art and goal instructions. Prints the initial room description. Enters a loop: decrements hunger randomly, checks if hunger reaches zero (triggers death), warns if hunger <30. Reads player input from cin, passes it to process\_command() for handling. Repeats until is\_running flag is false.
* process\_command(input): Normalizes input string. Matches input to known commands:
  + Movement commands: resolve direction alias, check for valid exit, update current\_room, call print\_room\_state().
  + Inventory commands: prints inventory contents, checks if empty.
  + Item interaction commands: validate presence in inventory/room, modify inventory or player state.
  + Combat commands: validate target character in room, resolve alias, apply damage to character, trigger enemy counterattack.
  + Dialogue commands: validate peaceful character in room, print greeting, optionally give gift.
  + Chest commands: validate chest space or item presence; add/retrieve items. Each command updates game state accordingly and prints output.
* print\_room\_state(moved): Prints full room description if first visit, or short description if revisiting. Prints available exits. Prints visible characters. Prints items on the ground. Prints chest contents if present.
* load\_rooms(filename): Reads rooms.txt line by line. Parses fields: id, description, size\_required, exit mappings. Constructs Room objects and populates rooms map.
* load\_items(filename): Reads items.txt line by line. Parses fields: id, description, location, damage, hunger\_restore, size\_change, aliases. Creates Item objects. Adds item IDs to initial items\_in\_rooms[location] list. Populates item\_alias\_map with each alias mapped to canonical id.
* load\_characters(filename): Reads characters.txt line by line. Parses fields: id, description, location, health, damage, drop\_items, aliases, peaceful, greeting, gift. Creates Character objects. Maps character IDs to characters\_in\_rooms[location]. Populates character\_alias\_map with aliases.
* load\_item\_aliases(filename): Reads alias mappings for items; updates item\_alias\_map.
* load\_character\_aliases(filename): Reads alias mappings for characters; updates character\_alias\_map.
* load\_required\_treasures(filename): Reads treasure IDs, trims whitespace, appends to required\_treasures vector.
* resolve\_item\_id(input): Normalizes input string (lowercase, strip punctuation). Returns canonical item ID from item\_alias\_map or empty string if not found.
* resolve\_character\_id(input): Same logic as resolve\_item\_id() but for characters.

**4.2 Player**

**File:** player.cpp, player.h

**Attributes:**

* health: integer (0–100)
* hunger: integer (0–100)
* size: string (e.g., "normal", "small", "large")
* base\_damage: integer (default 1)
* inventory: vector<string> of item IDs

**Functions:**

* get\_health(): Returns health.
* take\_damage(amount): Subtracts amount from health. Clamps health to 0 minimum.
* heal(amount): Adds amount to health. Clamps to 100 maximum.
* get\_hunger(): Returns hunger.
* change\_hunger(amount): Adds amount to hunger. Clamps within 0–100.
* get\_size(): Returns current size.
* set\_size(size): Assigns new size value.
* add\_item(id): Appends id to inventory vector.
* remove\_item(id): Removes id from inventory if exists.
* has\_item(id): Returns true if inventory contains id.
* get\_inventory(): Returns reference to inventory vector.
* show\_status(): Prints HEALTH and HUNGER bars using 20 segments filled or empty depending on value percentage.

**4.3 Character**

**File:** characters.cpp, characters.h

**Attributes:**

* id, description, health, damage, drop\_items (vector), peaceful (bool), greeting, gift\_item

**Functions:**

* get\_description(): Returns character description.
* get\_health(): Returns health.
* get\_damage(): Returns attack damage.
* take\_damage(amount): Subtracts amount from health; clamps to zero.
* is\_alive(): Returns true if health > 0.
* is\_peaceful(): Returns peaceful flag.
* get\_greeting(): Returns greeting string.
* get\_gift(): Returns gift\_item string.
* clear\_gift(): Sets gift\_item empty.
* load\_characters(filename): Populates character map from file.
* load\_aliases(filename): Loads aliases into alias map.
* resolve\_alias(input): Normalizes input, returns canonical ID.
* handle\_combat(enemy, enemy\_id, damage\_dealt): Applies damage to enemy. Checks death. If dead, prints death message, moves drop\_items into room items, removes character from room. If alive, triggers enemy counterattack.

**4.4 Item**

**File:** item.cpp, item.h

**Attributes:**

* id, description, damage, hunger\_restore, size\_change

**Functions:**

* get\_description(): Returns description.
* get\_damage(): Returns damage.
* get\_hunger\_restore(): Returns hunger\_restore.
* get\_size\_change(): Returns size\_change.
* load\_items(filename): Loads item data.
* load\_aliases(filename): Loads aliases.
* resolve\_alias(input): Normalizes input; returns canonical item ID.

**4.5 Room**

**File:** location.cpp, location.h

**Attributes:**

* id, description, size\_required, exits

**Functions:**

* get\_description(): Returns room description.
* get\_size\_required(): Returns required size.
* get\_next\_room(command): Returns destination room for command.
* add\_exit(command, dest): Adds exit mapping.
* get\_all\_exits(): Returns exits map.
* load\_rooms(filename): Loads rooms from file.

**4.6 Control**

**File:** control.cpp, control.h

**Functions:**

* normalize\_input(input): Lowercases and removes punctuation.
* normalize\_direction(input): Maps synonyms to canonical directions.

**5. Auxiliary Functions**

**5.1 Gradual\_Text**

**File:** gradual\_text.cpp

Defines cool\_text(text): Iterates characters; prints with flush; uses sleep\_until() on incremented steady\_clock timestamp.

**6. Data File Formats**

Each loader parses fixed-format text files corresponding to its class:

* rooms.txt: id|description|required\_size|exit1=dest1,...
* items.txt: id|description|location|damage|hunger\_restore|size\_change|aliases
* characters.txt: id|description|location|health|damage|drops|aliases|peaceful|greeting|gift
* item\_aliases.txt: id|alias1,alias2,...
* character\_aliases.txt: id|alias1,alias2,...
* required\_treasures.txt: list of item IDs

**7. Execution Flow**

1. setup(): loads all data.
2. Prints intro.
3. Prints starting room.
4. run(): loop input → process → update state → output.
5. Checks victory in safe room with all treasures.

**8. Design Justification**

* **OOP Structure:** Classes encapsulate data and behavior per game entity.
* **Data-Driven:** All content externalized for easy modification.
* **Input Normalization:** Handles synonyms, aliases, flexible commands.
* **Centralized Logic:** Game class coordinates class interactions.
* **No third-party dependencies:** Only standard C++ library headers used.

**9. Planned Extensions**

Future features: item-based puzzles, multi-turn dialogues, random encounters & special event triggers.