CSC111 Winter 2026 Project 1

Alexander Davydenko, Sophi Shu, Abhirve Munipalle

February 6, 2026

Running the game

Run the game by executing `python adventure.py` in the terminal. No additional module installations are required beyond the standard Python library.

Game Map

The game map consists of 7 locations connected as follows:

```
            [4: Robarts Library (Locked)] - [6: CSC111 Lecture Room]
                     |
 [2: Hallway] - [3: University College]
     |                  |
[1: Dorm Room]   [5: Bahen Centre]
                        |
                [7: Secret Server Room] (locked)
```

Starting location is: 1 (Dorm Room)

Game solution

Winning command list (one possible solution):

```
take t_card
go north
go east
go north
take usb_drive
go south
go south
take laptop_charger
go north
go north
go east
take lucky_mug
go west
go south
go west
go south
```

```
drop usb_drive
drop laptop_charger
drop lucky_mug
```

Lose condition(s)

Description of how to lose the game:

The player loses if they exceed the maximum number of moves without completing the objective. The move limit depends on difficulty:

- Easy mode: 40 moves
- Normal mode: 30 moves

Which parts of your code are involved in this functionality:

`adventure.py`, lines 292-297 in the main game loop checks if `game.moves >= game.max_moves` and sets `game.ongoing = False` when the condition is met.

Inventory

1. All location IDs that involve items in the game: 1, 4, 5, 6, 7
2. Item data:
   (a) For Item 1 (usb_drive):
   - Item name: usb_drive
   - Item start location ID: 4 (Robarts Library)
   - Item target location ID: 1 (Dorm Room)

   (b) For Item 2 (laptop_charger):
   - Item name: laptop_charger
   - Item start location ID: 5 (Bahen Centre)
   - Item target location ID: 1 (Dorm Room)

   (c) For Item 3 (lucky_mug):
   - Item name: lucky_mug
   - Item start location ID: 6 (CSC111 Lecture Room)
   - Item target location ID: 1 (Dorm Room)

   (d) For Item 4 (t_card):
   - Item name: t_card
   - Item start location ID: 1 (Dorm Room)
   - Item target location ID: 4 (Robarts Library) - used as a key

   (e) For Item 5 (server_room_key):
   - Item name: server_room_key
   - Item start location ID: 5 (Bahen Centre)
   - Item target location ID: 5 (Bahen Centre) - used as a key

   (f) For Item 6 (open_ai_api_key):
   - Item name: open_ai_api_key
   - Item start location ID: 7 (Secret Server Room)

- Item target location ID: 1 (Dorm Room) - secret ending trigger

3. Exact command(s) to pick up and drop items (inventory_demo):

```
take t_card
go north
go east
go north
take usb_drive
inventory
drop usb_drive
take usb_drive
```

4. Which parts of your code are involved in handling the `inventory` command:

   - File: `adventure.py`
   - Class: `AdventureGame`
   - Methods: `display_inventory()` (lines 137-143), `add_item_to_inventory()` (lines 133-135)
   - Main loop: Lines 427-428 handle the inventory command

Score

1. Briefly describe the way players can earn score in your game:

   Players earn points by returning items to their target location (Dorm Room, location ID 1). Each of the three required items awards different points:

   - usb_drive: 50 points when dropped in Dorm Room
   - laptop_charger: 30 points when dropped in Dorm Room
   - lucky_mug: 20 points when dropped in Dorm Room

   The first location where score can be increased is location 1 (Dorm Room), after collecting and dropping any of the required items.

   Commands leading to the first score increase:

```
take t_card
go north
go east
go south
take laptop_charger
go north
go west
go south
drop laptop_charger
```

2. scores_demo command list:

```
take t_card
score
go north
go east
go south
take laptop_charger
score
go north
go west
go south
drop laptop_charger
score
```

3. Which parts of your code are involved in handling the `score` functionality:
   - File: `adventure.py`
   - Class: `AdventureGame`
   - Method: `increase_score()` (lines 145-148)
   - Instance variable: `self.score` (initialized line 93)
   - Main loop: Lines 429-430 handle the score display command
   - Score increase logic: Lines 577-583 check if item is dropped at target location and awards points

General Menu Commands

The game supports the following general commands that can be used at any location:

1. `help` - Displays a list of all available commands and their descriptions
2. `look` or `l` - Shows the detailed description of the current location, including available exits and items
3. `inventory` or `i` - Displays all items currently in the player's inventory
4. `score` - Shows the player's current score and number of items returned
5. `map` - Displays a map of all visited locations and their connections. The current location is marked with "YOU ARE HERE"
6. `examine [item]` or `x [item]` - Shows detailed information about a specific item in the current location or inventory
7. `quit` - Exits the game (with confirmation prompt)

Movement Commands

Players can move between locations using directional commands:

- `go north` or `n` - Move north
- `go south` or `s` - Move south
- `go east` or `e` - Move east
- `go west` or `w` - Move west
- `go up` or `u` - Move up
- `go down` or `d` - Move down

Item Commands

- `take [item]` - Pick up an item from the current location
- `take` - If only one item is present, automatically picks it up
- `drop [item]` - Drop an item from inventory at the current location

Code Implementation

These commands are handled in `adventure.py`:

- Lines 241-251: Command aliases dictionary mapping shortcuts to full commands
- Lines 404-416: Alias expansion in input handling
- Lines 417-608: Main command processing logic in the game loop

Enhancements

1. Enhanced User Interface System
   - Brief description: A comprehensive UI overhaul including:
     - Directional cross showing available exits with location names
     - Item counter (X/3) replacing score in status bar
     - Move warning when 5 or fewer moves remain
     - Command aliases (n/s/e/w/u/d for movement, i for inventory, x for examine, l for look)
     - Visual formatting with borders, spacing, and organized sections
   - Reasons: This enhancement required significant code restructuring across multiple sections. The directional cross system involves parsing available commands, formatting dynamic brackets based on longest location name, and handling vertical (up/down) directions separately. The UI updates happen every turn and must dynamically adjust to the current location's connections. Implementation spans 50+ lines of code with complex string formatting and conditional logic.
   - Code involvement:
     - `adventure.py` lines 299-312: Status bar with items counter and move warning
     - `adventure.py` lines 332-378: Directional cross generation and display
     - `adventure.py` lines 241-251: Command aliases dictionary
     - `adventure.py` lines 404-416: Alias expansion in input handling
   - Enhancement demo commands:

     ```
     n
     s
     i
     map
     look
     ```

2. Examine Items Feature
   - Brief description: Players can examine items without picking them up using the 'examine' or 'x' command. Works for items in the current location or in the player's inventory. Shows detailed item descriptions.

- Reasons: Required implementing item search across both location items and inventory, handling multiple item aliases (mug, usb, charger, etc.), and integrating with the command system. The feature needed to be accessible both as a standalone command and with arguments. Code spans multiple sections including menu handling, alias expansion, and item lookup logic (approximately 60 lines).
- Code involvement:
  - `adventure.py` lines 455-474: Examine command in menu handler
  - `adventure.py` lines 514-529: Examine with argument handler
  - Item aliases integrated in lines 464, 470, 520, 526
- Enhancement demo commands:

  ```
  examine t_card
  x t_card
  go north
  go east
  go south
  examine laptop_charger
  ```

3. Locked Locations Puzzle System
   - Brief description: Players must find specific keys to unlock certain locations. Robarts Library requires the t_card, and the Secret Server Room requires the server_room_key. Without the required key, players cannot enter these locations. Once unlocked, locations remain accessible.
   - Reasons: This is a complex puzzle system requiring:
     - Modified Location class to support locked state and key_id
     - Item class extension to track which items serve as keys
     - JSON data structure additions for locked locations
     - Movement command logic to check inventory for keys
     - Feedback system to inform players why they can't enter
     - Permanent state changes once unlocked

     Changes span multiple files (`game_entities.py`, `adventure.py`, `game_data.json`) and involve data structure modifications, inventory checking, and conditional movement logic.
   - Code involvement:
     - `game_entities.py` lines 34-35, 54-57: Location class lock attributes
     - `adventure.py` lines 109-110: Loading lock data from JSON
     - `adventure.py` lines 489-510: Lock checking logic in movement handler
     - `game_data.json`: Robarts Library (id 4) and Secret Server Room (id 7) with locked/key_id fields
   - Enhancement demo commands:

     ```
     go north
     go east
     go north
     (fails - locked)
     go south
     go west
     go south
     take t_card
     ```

```
go north
go east
go north
(succeeds - unlocked with key)
```

4. Secret Ending Path

   - Brief description: Players can discover a secret ending by finding the open_ai_api_key in the hidden Secret Server Room and returning it to their Dorm Room. This triggers an alternate ending where the player uses AI to complete the assignment but gets caught for academic dishonesty, resulting in a 0% grade and suspension.
   - Reasons: This feature required:
     - Adding a new hidden location (Secret Server Room) accessible only via locked door
     - Creating a new quest item with unique behavior
     - Implementing alternate ending condition that checks before win condition
     - Integrating with the item drop system to trigger the ending

     The implementation involves careful ordering of checks (secret ending before win condition), and integration with existing systems. Code spans multiple files and includes narrative design.
   - Code involvement:
     - `adventure.py` lines 219-225: `check_secret_ending()` method
     - `adventure.py` lines 586-620: Secret ending trigger and narrative
     - `game_data.json`: Location 7 (Secret Server Room) and item 5 (open_ai_api_key)
   - Enhancement demo commands:

     ```
     go north
     go east
     go south
     take server_room_key
     go down
     take open_ai_api_key
     go up
     go north
     go west
     go south
     drop open_ai_api_key
     ```

5. Difficulty Selection & Achievement System

   - Brief description: At game start, players choose between Easy (40 moves) or Normal (30 moves) difficulty. Players who complete the game in under 20 moves earn the "Speedrunner" achievement shown on the victory screen.
   - Reasons: Required implementing:
     - Pre-game menu system with input validation
     - Dynamic move limit adjustment based on difficulty
     - Move tracking and comparison at win condition
     - Enhanced victory screen with conditional achievement display

     The feature integrates with the intro sequence, game state management, and win condition checking.
   - Code involvement:

  - `adventure.py` lines 267-283: Difficulty selection menu
  - `adventure.py` lines 640-643: Speedrunner achievement check
  - `adventure.py` line 287: Achievement promotion in intro
- Enhancement demo: Complete the game in under 20 moves to see the achievement

6. Dynamic Map Display

- Brief description: The 'map' command shows all visited locations with their connections. The current location is marked with "YOU ARE HERE". Connections show location names if visited, or "?" if unexplored.
- Reasons: Implementation required:
  - Tracking visited state for all locations
  - Iterating through location connections
  - Conditional display based on visit status
  - Marking current position

  The feature provides valuable navigation assistance and integrates with the location visiting system.
- Code involvement:
  - `adventure.py` lines 150-171: `display_map()` method
  - `adventure.py` lines 433-434: Map command handler
  - `game_entities.py` line 53: visited attribute in Location class
- Enhancement demo commands:

```
map
go north
map
go east
map
```

7. Smart Item Interaction

- Brief description: Players can type just 'take' without specifying an item. If there's only one item in the location, it's automatically picked up. If multiple items exist, the game prompts the player to choose.
- Reasons: Simple conditional logic that checks item count and either auto-selects the single item or prompts for selection. Implementation is straightforward but improves user experience significantly.
- Code involvement:
  - `adventure.py` lines 534-545: Auto-take logic
- Enhancement demo commands:

```
take
(automatically takes t_card)
```

8. Quit Confirmation

- Brief description: When the player types 'quit', the game asks "Are you sure you want to quit? (yes/no)" to prevent accidental exits.
- Reasons: Simple confirmation prompt with yes/no validation. Straightforward implementation that prevents frustrating accidental quits.

- Code involvement:
  - `adventure.py` lines 449-454: Quit confirmation handler
- Enhancement demo: Type 'quit' then 'no' to see confirmation