CSC111 Winter 2025 Project 1

Bryan Yee
Deepankar Garg

February 12, 2025

Running the game

This project does not have any extra libraries to install. To run the game, simply run `adventure.py`

Game Map

```
 -1  1 -1 -1
  8  2  3  4
  6  9  5 -1
 -1  7 -1 -1
```

The starting location is: 1

Game solution

List of commands: [”go south”, ”go south”, ”go south”, ”talk to professor”, ”go north”, ”go west”,”pickup charger”,”go east”,”go north”,”go west”,”talk to group member”,”go east”,”go north”,”submit project”]

Lose condition(s)

Description of how to lose the game:
The lose condition of this game is when the player runs out of moves (by default, 14 moves). A move occurs whenever the player uses the ”go” command to enter a new location. However, any commands used within a room do not count as moves.

List of commands:
[”go south”, ”go east”, ”go east”, ”go west”, ”go west”, ”go south”, ”go east”, ”go west”, ”go south”, ”go north”, ”go west”, ”go east”, ”go north”, ”go west”, ”go east”]
In this example, the player attempted to visit every location, but ran out of moves in the process resulting in the game being a loss.

Which parts of your code are involved in this functionality:
In `adventure.py`, the use_command() is called everytime the player chooses a command. If the ”go” command is used, then the number of moves stored in player object will be decreased by one. In the main game loop, also located in `adventure.py`, there is an if loop checking to see if the player has any remaining moves after a command is called which decides whether the game will continue or not.

Inventory

Note: Not all items can be used at any given moment, and can only be used in certain locations.

1. All location IDs that involve items in the game:

   (a) Location ID, 1: Dorm
   - Item 1: Notebook
     - Pickup command: "pickup notebook"
     - Use command: "check notebook"
       - ∗ Used anywhere
       - ∗ Using notebook will inform the player on the objective of the game

   (b) Location ID, 2: Courtyard
   - Item 2: Scrap paper
     - Pickup command: "pickup scrap paper"
     - Use command: "throw out scrap paper"
       - ∗ Used in Location 2, Courtyard
   - Item 3: Cash
     - Pickup command: "throw out scrap paper"
       - ∗ Can only be called once player has Item 2, Scrap paper
     - Use command: "buy coffee"
       - ∗ Used in Location 4, Cafe

   (c) Location ID, 3: Library
   - Item 4: Textbook
     - Pickup command: "check bookshelves"
     - Use command: "exchange textbook for project tips"
       - ∗ Used in Location 4, Cafe

   (d) Location ID, 4: Cafe
   - Item 5: Coffee
     - Pickup command: "buy coffee"
       - ∗ Will only give player item if the player has Item 3, Cash
     - Use command 1: "drink coffee"
       - ∗ Used anywhere
       - ∗ Gives player 2 extra moves
     - Use command 2: "give half asleep student coffee"
       - ∗ Used in Location 8, CS Lounge

   (e) Location ID, 5: Classroom 2
   - Item 6: Sticky Note
     - Pickup command: "exchange textbook for project tips"
       - ∗ Can be called once player calls "talk to upper year student", in Location 5, Classroom
       - ∗ Will only give player item if the player has Item 4, Textbook

   (f) Location ID, 6: Classroom 1
   - Item 7: Laptop Charger
     - Pickup command: "pickup laptop charger"

(g) Location ID, 8: CS Lounge
  - Item 8: USB
    - Pickup command: "pickup laptop charger"
  - Item 9: Lucky Mug
    - Pickup command: "give half asleep student coffee"
      - Can be called once player calls "talk to half asleep student", also in Location 8, CS Lounge
      - Will only give player item if the player has Item 5, Coffee
  - Item 10: Comic Book
    - Pickup command: "search bookshelves"
    - Use command: "read comic book"
      - Used in Location 1, Dorm

2. Which parts of your code (file, class, function/method) are involved in handling the `inventory` command:
   The Player class in `game_entities.py` stores the attribute for the player's inventory. This attribute is an array of Item classes. When the `inventory` command is called in the main game loop, it will call `player.show_inventory()` and print the list of items in the player's inventory into the console.

Score

1. Briefly describe the way players can earn scores in your game. Include the first location in which they can increase their score, and the exact list of command(s) leading up to the score increase:

   Various puzzles, some of which are mandatory, will reward the player with score upon completion.

   Score from throwing away scrap paper: 1000 Score

   - Go to the Courtyard from the starting location
   - Pickup scrap paper
   - Throw scrap paper to get the item, Cash (+1000 score)

   Commands: ["go south", "pickup scrap paper", "throw out scrap paper"]

   Score from Sticky Note: 2000 Score

   - Go to the Courtyard, and then the library
   - Search the bookshelves for a textbook (+1000 score)
   - Go to the Classroom
   - Talk to the upper year student
   - Exchange the textbook for project tips, to get the sticky note (+1000 score)

   Commands: ["go south", "go east", "search bookshelves", "go east", "talk to upper year student", "exchange textbook for project tips"]

   Score from Comic Book (This puzzle decreases the player's score, so the player will have to learn to avoid doing these steps): -1000 score

- Go to the Courtyard, and then the CS Lounge
- Search the bookshelves for a comic book
- Go back to the courtyard, and then the dorm
- Read the comic book (-1000 score)

Commands: ["go south", "go west", "search bookshelves", "go east", "go north", "read comic book"]

Score from Lucky Mug: 2000 Score

- Repeat steps in the scrap paper puzzle to get cash
- Go to the library from the courtyard, and then the cafe
- Buy coffee with the item, cash
- Go back to the library, then courtyard, and then CS Lounge
- Talk to the half asleep student
- Give the student the item, coffee and recieve the item, lucky mug (+2000 score)

Commands: ["go south", "pickup scrap paper", "throw out scrap paper", "go east", "go east", "buy coffee", "go west", "go west", "go west", "talk to half asleep student", "give half asleep student coffee"]
Note: When the player obtains the coffee, they have to decide whether they want to save it in order to get the lucky mug, or drink the coffee themselves to obtain an extra 2 moves.

Score from Laptop Charger (mandatory to complete the game): 4000 Score

- Go to the Courtyard, and then the class building, then the prof office
- Ask the prof to unlock Classroom 1 (+1000 score)
- Go back to the class building, and then Classroom 1
- Pickup the laptop charger (+3000 score)

Commands: ["go south", "go south", "go south", "talk to prof", "go north", "go west", "pickup laptop charger"]

Score from USB (mandatory to complete the game): 3000 Score

- Go to the Courtyard, and then the CS Lounge
- Talk to the group member to obtain the USB (+3000 score)

Commands: ["go south", "go west", "talk to group member"]

2. Copy the list you assigned to `scores_demo` in the `project1_simulation.py` file into this section of the report:

3. Which parts of your code (file, class, function/method) are involved in handling the `score` functionality:
   In the `game_data.json` file, where the list of available commands are listed out, each command that would result in a change of score has a parameter indicating how much score the player will earn upon calling the command. In `adventure.py`, where the main game loop is executed, calling a command (the use_command() method) will check to see if the chosen command has a score change or not. If it does, then it will update the player's score attribute using the ******** method.

Enhancements

1. Simple and Complex Puzzles

   - The game has various simple and complex puzzles that reward the user with score as described in the *Score* section of this report. The complex puzzles (Laptop Charger and Sticky Note) all require multiple steps, such as buying items and moving around to complete. All the other puzzles are considered simple as they only have 1-3 steps to complete.
   - Complexity level: mixed (depended on the command)
   - We altered the default code to be able to create functional commands almost completely with `game_data.json` and the Command class found in `game_entities.py`. This would allow for as little hard coding as possible when selecting commands. In addition, it was difficult to design puzzles that the player would have to strategically decide which puzzles are the best to complete in the given amount of moves.

2. The "buy" Command Type

   - The buy command allows the player to exchange an item in their inventory for another item. Certain locations will have a buy command that are either available by default, or unlocked upon calling another command (ex. talking to a student). Note: Unlike the "go" command, not all "buy commands" are labeled as "buy [item]". Instead, all "buy commands" are characterized by requiring 1 item, and rewarding another.
   - Complexity level: medium
   - This enhancement is of medium difficulty because although it functions similar to a "pickup" item command, it requires extra conditions (ex. checking if the player has the required item) to check if the player is able to use this command. What made this command difficult to implement, was not the functionality but rather designing how the game data would store the command's data.

3. The "talk" Command Type

   - The talk command will print some dialogue, leading to further events occuring. The availability of the talk command is dependant on the player's current location. Using the talk command will typically result in another command being unlocked (like a buy command)
   - Complexity level: low
   - This enhancement is of low difficulty because it will simply print out some text from `game_data.json` into the console. Similar to the buy command, the main difficulty was creating a way to define the commands with very little hard coding. For the talk command, we had to create a way in which it would unlock other commands.

4. The Player Class

- The Player class stores the data for the player's inventory, score, and moves left. The class is found in `game_entities.py`
- Complexity level: low
- This enhancement is of low difficulty because it simply stores variables attached to the player and provides basic methods such as displaying the inventory.