Text Adventure Game

Give yourself a pat on the back for making it this far! As we wrap up this course, it's time to show off your programming skills with a creative project. You'll be designing and coding a simple text-based adventure game.

The game will consist of a series of interconnected rooms, each filled with various items and events. The player will be given some flavor text (a description of their surroundings and the situation they're in) as well as a list of commands or actions they can take. They will interact with the game by entering these text commands into the terminal.

Instructions

Choose a Game Concept

Decide on the setting and general story of your game. It should be fairly simple. The game should involve navigating through different locations and interacting with the environment. Examples might be escaping a haunted mansion, surviving on a deserted island, or navigating through a mystical forest. The story is just an excuse for the gameplay, so don't overthink this part.

Design the Game Map

Plan out your world map, including the locations, how they connect, and what items can be found where. You may want to do this by physically sketching it out on a piece of paper. Include 6-12 unique rooms, each with their own events to interact with or items to collect. Have fun and make it interesting! Look back on the concepts you've learned and think about creative ways to implement them.

Implement Core Game Functions

Once you've finished your design, start implementing the game in Python. Reach out to the tech coach and other students for help and feedback as you work. Start by implementing the following functions.

`exposition()`: This function returns a string consisting of a few lines of expositional text establishing the initial game setup or story.

create_world(): This function sets up the initial state of the world. This involves initializing a dictionary called “world” that establishes the game state (whether the player is currently playing, won the game, lost the game, or quit the game), the player’s starting location and inventory, and the world map, which is a list containing each of the rooms. Each room is a dictionary which includes a name, a description, a list of neighboring locations, and any items that can be found there. For example, your game world might look something like this:

```
world = {"status"   : "playing",
        "player"   : {
              "location"  : "foyer",
              "inventory" : []
                },
        "world_map" : [
           {
           "location" : "foyer",
           "about" : "You find yourself in a dimly lit foyer. A
           grandfather clock sits in the corner, ticking away with a
           haunting rhythm. You find a key on the ground.",
           "neighbors" : ["hallway"],
           "items" : ["key"],
           },
           {
           "location" : "hallway",
           etc...
           },
         ]
       }
```

render(world): This function takes in the current world state and returns a string that describes it to the player. This might involve finding the player’s current location, finding that location in the world map, and looping through the player’s inventory to determine what information needs to be rendered.

get_options(world): This function takes in the current world state and returns a list of valid commands the player can choose at that time. This might include “north”, “south”, “east”, and “west” commands so the player can choose where to go. These are just suggestions, and you are free to use any approach that works for your game. For other commands, you might use conditional statements to see where the player is at and what items they have, among other things. For example:

```
if "MAP" in world["player"]["inventory"]:
    options.append("MAP")
```

update(world, command): This function takes in the current world state and a command, and then updates the world based on that command. It should also return a message about what happened. Here’s an example:

```
loc = world["player"]["location"]
if command == "QUIT":
    world["status"] = "quitting"
    return("GAME OVER. Play again soon!")
elif world["player"]["location"] == "foyer" and command == "NORTH":
    if "KEY" not in world["player"]["inventory"]:
        print("\nThe door seems to be locked. It won't budge.")
    else:
        print("\nYou try the key. It works!")
```

render_ending(world): This function returns a message to be displayed at the end of the game, depending on the status of the world. If the status is “playing”, you should still be in a while loop and not render an ending. If the status is “win”, “lose”, or “quitting”, you’ll want to give the player an appropriate ending. You might include multiple losing scenarios, and maybe a “golden ending” winning scenario where the player wins the game while also fulfilling certain additional conditions.

choose(options, inventory): This function takes in a list of options and the user’s inventory and asks them to choose an action. It should keep asking until a valid option is chosen, and then return that option. You may also want to print the player’s inventory for their convenience.

```
    print("COMMANDS:")
    for i in range(len(options)):
        print(options[i])
    if inventory != []: # if inventory is not empty
        print("INVENTORY:")
        for i in range(len(inventory)):
            print(inventory[i])
    while True:
        command = input("What do you want to do? ")
        if command in options:
            break
        else:
            print("Invalid command.")
    return command
```

main(): This function is the main loop of the game. It should use all the other functions to create the world, get the user's choice, update the world, and repeat as long as the game is playing. It should look something like this:

```
def main():
    print(exposition())
    world = create_world()
    while world["status"] == "playing":
        print(render(world))
        options = get_options(world)
        inventory = world["player"]["inventory"]
        command = choose(options, inventory)
        print(update(world, command))
    print(render_ending(world))
    input("Press any key to continue.")
    print()
    main()
```

Criteria

Interactivity

The game should take input from the user to navigate through the world and interact with the environment. The game should maintain a state that changes based on the user's actions. The game should clearly communicate the current state to the user, and the user should always know which actions are available to them. There should be a clear end point where the user either wins or loses. After the game is over, there should be an option to play again.

Complexity

The world should contain at least six rooms with unique events and there should be at least three loot items, but you are encouraged to add more if your game calls for it. Challenge yourself and implement concepts that you've learned so far. There should be multiple valid sequences of actions that lead to winning the game.

Error Handling

The game should handle invalid commands gracefully. The player should not be able to crash the game by giving unexpected or invalid inputs. Test your game rigorously and try every option in every scenario to make sure your code behaves as expected. Let other people playtest your game and try to "break" it.

Code Quality

The code should be well-structured and easy to understand, with clear function and variable names and comments explaining anything complex. The game itself should also be easy to understand. Let other people play your game and watch how they interact with it. Don't tell them how to do anything while you watch them play; your game should explain how to play it.