Assignment 8 Worksheet | Due November 10 at 11:59PM

Planning Worksheets

Edit this file and save it as a PDF and submit a single .pdf file to the assignment drop box. Both you and your partner must submit a copy of this completed PDF to Brightspace. The file you submit should be named `comp1405_f23_###_###_assignment_08.pdf` with `###` replaced by the last three digits of each partner's Student ID number. If you are working alone, only put your own Student ID number.

Brightspace. Step 0: Features

What features have you and your partner selected? Remember to choose one from each category.

1. **Category 1:** Loading Game Settings from a File
2. **Category 2:** Number Anomaly

Step 1: User Stories

For both of your selected features, write **one to two User Stories each** which express the main value of the features. Follow the format seen in class: "As a **<role>,** I want to **<action>,** so that I can **<motivation>**"

User Story 1 (For Category 1):

- As a **user,** I want to **load game settings from a file,** so that I can **adjust the game's difficulty to my liking.**

User Story 2 (For Category 1):

- As a **user,** I want to **have a user-friendly game that allows me to modify game settings with a command line argument,** so that I can **change the game settings with a simple command line argument rather than modifying Python code. If no command line argument is provided use the default settings.**

User Story 1 (For Category 2):

- As a **user,** I want to **come across number anomalies in the game, where numbers in item descriptions change,** so that I can **enjoy a compelling and unpredictable gaming experience that keeps me engaged and excited about what is to come.**

Step 2: Requirements & Constraints

Carefully consider your feature selections. Review the code, the provided documentation, and the feature description to write a list of required functionality and constraints that must be considered. Consider cases where the feature might not work as intended, changes to the data to show your feature, or techniques you cannot use. As an example, one constraint for all features is that **the Duty.py module <u>cannot</u> be modified.**

Category 1

"As a **user**, I want to **load game settings from a file**, so that I can **adjust the game's difficulty to my liking**."

❑ Create a command line argument to specify a text file for loading game settings.
❑ Ensure the game is fully functional without a command line argument, utilizing default settings from the Duty module's 'init()' function.
❑ Read and apply the game settings from the specified text file.

❑ Effectively manage situations where the specified text file is unavailable or contains errors, ensuring it doesn't crash or disrupt the gameplay
❑ Do not modify the Duty.py module.

"As a **user,** I want to **have a user-friendly game that allows me to modify game settings with a command line argument,** so that I can **change the game settings with a simple command line argument rather than modifying Python code."**

❑ Implement a command line argument for users to modify game settings easily.
❑ Keep the process of changing settings user-friendly and straightforward.
❑ Ensure users can adjust game settings without needing to access and modify Python code.
❑ Do not modify the Duty.py module.

Category 2

"As a **user,** I want to **come across number anomalies in the game, where numbers in item descriptions change,** so that I can **enjoy a compelling and unpredictable gaming experience that keeps me engaged and excited about what is to come."**

❑ Implement number anomalies that randomly alter item descriptions containing numbers.
❑ Ensure the altered numbers are either higher or lower by one at random, adding variety and challenge to the gameplay.
❑ Ensure that the implementation of number anomalies does not break the game or make it impossible to play.
❑ Detect item descriptions with numbers accurately to apply the number anomaly.
❑ Conduct testing to verify that number anomalies work as intended without introducing unintended/new issues.
❑ Efficiently integrate the logic for number anomalies into the existing game code.
❑ Do not modify the Duty.py module.

Step 3: Prioritize

While you may not have many User Stories, with your partner, determine the best order to complete these features and write the order below. The highest priority will be at the top in position 1.

Prioritized User Stories

1. Load game settings from a file
2. Create and add number anomalies
3. Have a user-friendly game that allows modifications to game settings with a command line argument

For each User Story, justify why you and your partner prioritized it here. **Be brief – one to two sentences maximum.**

1. This user story is prioritized first because it provides players with the necessary functionality of actually loading game settings from a file. It also enables players to customize their game experience, which is a significant value addition.

2. This user story is prioritized second because although it adds variety and excitement to the game, it relies on the first user story of category 1 to provide customizable game settings.
3. This user story is prioritized last because while it enhances user-friendliness by allowing settings modification through a command line argument, it is considered of slightly lower priority compared to the other user stories, which directly impact gameplay variety and functionality.

Step 4: Task List

For each user story, write a **task list**. In real work, tasks could be as large as the user stories, but in this class we will work at a smaller scale. For each user story, write a **list of tasks** which cover everything that your team will need to do to write the code, make sure it works, and submit in Assignment 9. Review the code in **RunGame.py** to see what steps need to be accomplished to do similar work to your features. Assume that you will be starting with the **RunGame.py** code and do not need to include tasks that are already included there. Include any functions that you will need to review or likely need to use in the task description. This is **not** an algorithm; it is a checklist for you and your partner.

Category 1 Task List(s)

1. Implementing Command Line Argument: Create the logic to handle command line arguments that specify the text file for loading game settings.
2. Read Default Settings: Ensure that the game works and does not produce errors even if no command line argument is provided, by reading and applying default settings from the Duty module's 'init()' function.
3. Read and Apply Game Settings: Implement the code to read and apply the game settings from the specified text file.
4. Manage Possible Errors: Manage/account for situations where the specified text file is unavailable or contains errors, ensuring that the game does not crash or disrupt gameplay.
5. Test and Debug: Thoroughly test the implementation to verify that game settings can be loaded from a file and that they are applied correctly. Debug any issues that may come up.
6. Submission: Prepare code for submission in Assignment 9, ensuring it aligns with the assignment requirements and builds on the **RunGame.py** code while not modifying the Duty.py module.

Category 2 Task List(s)

1. Implement Number Anomaly Logic: Develop the logic for number anomalies (i.e. anomalies that randomly alter item descriptions containing numbers, making sure the change is either higher or lower).
2. Item Description Detection: Create a method to accurately detect item descriptions that contain numbers and apply the number anomaly.
3. Integration with Existing Code: Efficiently integrate the logic for number anomalies into the existing game code, considering factors like room selection and anomaly creation.
4. Testing: Conduct thorough testing to verify that number anomalies work as intended without introducing unintended or new issues that could disrupt the gameplay.
5. Submission: Prepare code for submission in Assignment 9, ensuring it aligns with the assignment requirements and builds on the **RunGame.py** code while not modifying the Duty.py module.

Assignment 8 Worksheet | Due November 10 at 11:59PM

Team Worksheet

Finally, this section is here to keep your team on track. It will receive full marks if you are registered in a group on Brightspace, and a reasonable effort was made to respond to each point. If you are working individually, you will not receive marks for this section.

Team Member Names: **1.** Abaan Noman **2.** Rahul Rodrigues
What days are you planning to meet to work together to code Assignment 9, in-person or online? We are planning to work together online to code Assignment 9 starting on Saturday, November 11. We will work together on the coding part on Saturday, Sunday, Monday, and if needed, Wednesday after our afternoon classes.
How are you planning to pair-program together (e.g., screen sharing, together in-person, VS Code Live Share)? For days other than Wednesday (the only day we can meet in person), we are planning to pair-program when needed by using VS Code Live Share while in a Discord call so we can write, edit, and discuss our code at the same time.
How long do you each anticipate working on the code for this assignment? We are currently anticipating working on this assignment for around 8 hours total, which would be 4 hours per person (2 hours combined each day and adjusting as needed).
Who will be submitting this PDF and who will be submitting the Assignment 9 Python file? Both of us will be submitting this PDF and the Assignment 9 Python file individually, as stated in the assignment requirements.

Remember to submit your responses as a PDF on Brightspace and ensure that **both group members have submitted.** You **must** have registered your group before the group registration deadline in order to receive marks for this assignment if you are working in a group.

Assignment 8 Worksheet | Due November 10 at 11:59PM

BONUS: Function Planning

This section is optional, but can provide **10% bonus marks**, up to a **maximum of 100%** on this assignment to make up for other lost marks, if completed. Bonus marks will be provided if an attempt is made to answer these questions on either a per-task or per-feature basis, whether working individually or as a team. Total coverage of all questions across all tasks is not strictly required, but aim to complete this as best as you can to better coordinate with your partner and have a clear picture of the work required for Assignment 9.

For each task in your task lists, try to write out and plan the following:

1. What functions from the **Duty.py** module will you need to use?
   The functions we will need to use from the **Duty.py** module are:

- display(): Displays in-game information, such as the current time, cameras, and active anomalies.
- init(anomalies, rooms, room_items, **settings): Initializes the game, including anomalies, rooms, and game settings.
- handle_input(clear=True): Handles user input and interprets commands.
- update(): Updates the game and checks if the game should continue or end based on conidtions such as too many anomalies, etc.
- rooms_without_anomalies(): Provides a list of room names without activated anomalies.
- number_of_anomalies_to_create(): Finds the number of anomalies to create based on probabilities and time since the last anomaly was checked.
- register_anomaly(name): Registers a new anomaly in the game.
- add_anomaly(name, room, modified_item_list): Adds an anomaly to a room, modifying the list of items if there is no current anomaly in the room.
- get_rooms(): Returns a list of all available rooms in the game.
- get_random_unchanged_room(): Returns a random room without an active anomaly.
- add_room(room_name, room_items): Adds a new room to the game, avoiding duplicates.
- get_room_items(room): Obtains a copy of the list of items in a specific room.
- get_random_anomaly(): Selects a random anomaly for the game.
- set_setting(setting, value): Allows adjustment of game settings during gameplay.

2. What functions from the **RunGame.py** file can you, and must you, modify or use as an example?
   The functions that can, and must be modified or used are:

- main(): The main function sets up and runs the game loop, making it the most important starting point for integrating your features (will be used for the command-line argument for loading game settings from a file and use default settings if it is not provided).
- add_rooms(): A helper function adding rooms to the game, useful for modifying game data.
- register_anomalies(): Another helper function registering possible anomalies, providing guidance on incorporating new anomalies (will be used for our number anomalies).
- create_anomaly(): A helper function controlling the flow for creating anomalies (will be used for our number anomalies).
- missing_item(room): A function that removes a random item from a room, demonstrating manipulation of room data (will also play a role in our number anomalies).

- item_movement(room): A function that rearranges two items in a room, showcasing more complex data manipulation (to test and implement number anomalies/game settings).

3. What functions should you write to cleanly separate our code into small pieces?

a. What could you name the function?
b. What is that function's one purpose?
c. What parameters will that function accept, and what data type are they?
d. Will the function return anything, and if so, what data type will it be and how will that return value be used?
e. What would an example of using this function look like?

Answer:
For Loading Game Settings from a File (Category 1):
a. Function Name: load_settings_from_file
b. Purpose: Read game settings from a specified text file.
c. Parameters: file_path (string): The path to the text file containing game settings.
d. Return Type: dict (or an object that represnts game settings).
e. Example: settings = load_settings_from_file("path/to/settings.txt")

For Game Settings (Category 1):
a. Function Name: modify_game_settings
b. Purpose: Modify game settings based on user input (through a command-line argument).
c. Parameters: command_line_args (list of strings): Command line arguments provided by the user.
d. Return Type: None (Settings will be modified directly).
e. Example: modify_game_settings(["--difficulty", "easy"])

For Implementing Number Anomalies (Category 2):
a. Function Name: implement_number_anomalies
b. Purpose: Implement the number anomalies to item descriptions containing numbers.
c. Parameters: item_descriptions (list of strings): The descriptions of items in a room.
d. Return Type: list (modified item descriptions).
e. Example: modified_items = implement_number_anomalies(["42\" TV Playing Golf", "Black Leather Sofa", "Circular Metal Coffee Table", "Wooden Bookshelf with 3 Shelves"])

Some tasks may not be appropriate for these questions. Try to come up with similar questions for your tasks.

For each feature, design a small algorithm by writing either a flowchart or a simple text algorithm which expresses the flow of events without worrying about the Python syntax.

Feature 1: Loading Game Settings from a File
Text algorithm:
Start
Read the command line arguments.

If a text file path is provided:
Read and go through the game settings from the specified text file.
Apply the loaded game settings to the game.
If no text file path is provided or an error occurs:
Use default game settings from the Duty module.
End

Feature 2: Number Anomalies
Algorithm:
Start
For each item description in the room:
Check if the item description contains a number.
If it does, randomly decide whether to increase or decrease the number by one.
Apply the change to the item description.
Return the modified list of item descriptions.
End

Write your responses either here or on later pages.