**COMP 1005B/1405ABC**

# Assignment #8

## Submission Deadlines & Requirements

**This is a group assignment.** To receive full marks, you must form a group with another member of the class or indicate that you would like to be randomly assigned to a group by filling out the "**Assignment 8 & 9 Group Registration**" quiz on Brightspace before **Sunday, November 5th at 11:59PM.**

For this assignment, you will **each** be **submitting** a **single .pdf** file containing a copy of your group's responses to the questions by downloading and working from the .docx template file **comp1405_f23_assignment_08.docx** which is available on Brightspace and must be submitted with the name:

`comp1405_f23_###_###_assignment_08.pdf`

Where ### is replaced by the last three digits of each of your group member's student ID numbers, or just your own ID number if working individually. If you are working individually **and acknowledge that you will not receive full marks**, you must **still register on Brightspace** before the registration deadline indicating this.

Officially, the **due date** for this assignment has been set for **Friday, November 10th, 2023, at 11:59 pm EST**, but it is important to note that any late submissions will still be accepted **without penalty** until **Sunday, November 12th, by 11:59 pm EST**.

You have been provided with two Python files: **Duty.py** and **RunGame.py.** You are expected to read **RunGame.py** to support this assignment and to prepare for Assignment 9 where you will modify that code by adding features that you plan here, but you will **not** be submitting these Python files for Assignment 8.

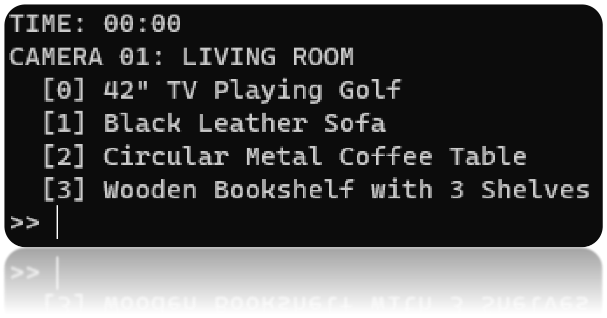
## Assignment Overview

This assignment does not require writing any Python code but instead prepares you for Assignment 9 where you will program new features for the game you have been provided. In Assignment 8, you and your partner will review the description for the provided project that has a relatively large and complex codebase (**Duty.py**) which you are **not** expected, required, or encouraged to read. Only read this code if you would like to learn more advanced Python concepts if you have prior experience programming, otherwise, **restrict yourself to this document for information to avoid confusion**. There is a lot of information here - this assignment is about reading, understanding, planning, and collaborating.

First, take time to read and understand the game that was provided by running the **RunGame.py** file. Instructions about the game are on the next page. Note: While this is not an endorsement of the game, this game was directly inspired by the game *I'm on Observation Duty* by developer *Notovia*. Note: Any knowledge of the source game is **not** required, and all information required is contained in this specification. Then, you will select new features which you will be adding in Assignment 9 and breaking them down into requirements and tasks. **Note: A video recording of the gameplay and overview of the provided RunGame.py has been provided as an extra resource on Brightspace alongside this assignment specification.**

## Game Rules

The game you were provided with is named "*I Am on Duty Watching Changes to Rooms*". In this game, the player is tasked with watching security cameras and looking for changes to those rooms. In our case, each camera is represented as an **ordered list of items**. The player can switch cameras by typing **n,** or **next**, to see the next camera, or **p** or **prev** to switch to the previous camera.



```
TIME: 00:00
CAMERA 01: LIVING ROOM
  [0] 42" TV Playing Golf
  [1] Black Leather Sofa
  [2] Circular Metal Coffee Table
  [3] Wooden Bookshelf with 3 Shelves
>>
```

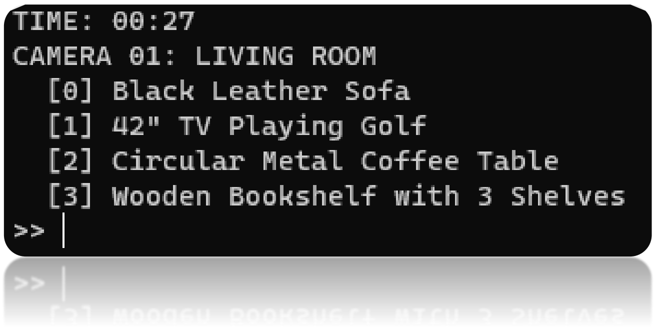
◀ **The in-game time**
◀ **The camera number and current room**
◀ **The list of items in a room**

The goal is to memorize the list of objects in the room and look for **Anomalies**. An **anomaly** is any change to the room's item descriptions or the room itself. For example, after a few in-game minutes, the list of items in the **LIVING ROOM** might be missing an item – a "**Missing Item Anomaly**", or the items might be re-arranged, an "**Item Movement Anomaly"**. A third anomaly has also been provided – a "**Camera Malfunction Anomaly"**, the camera cannot be switched to (i.e., it is skipped when typing next or prev). Try running the game and seeing an example of this in action.

**Notice that item 0 and item 1 have swapped.**

**Uh-Oh!** ▶
▶
**The Living Room has an "Item Movement" Anomaly!**

**We will need to report the anomaly, or we will lose!**



```
TIME: 00:27
CAMERA 01: LIVING ROOM
  [0] Black Leather Sofa
  [1] 42" TV Playing Golf
  [2] Circular Metal Coffee Table
  [3] Wooden Bookshelf with 3 Shelves
>>
```

Each room can only have one anomaly active at a time. When there are too many anomalies active at one time across all rooms, the player loses. Players receive a warning when there is only one anomaly left before losing.

The goal of the player is to avoid having too many anomalies active by the end of the game. To prevent this, the player can **Report** an anomaly by typing **r** or **report**. They then type the room that the anomaly is in and the type of anomaly they suspect is in the room. If the anomaly was active in the room, then it is removed.

To adjust difficulty, the game's settings – such as how fast the in-game clock ticks forward, how many anomalies must be active before losing the game and more, can be adjusted in the `main()` function of RunGame.py. All the settings are described on a later page.

**In Summary:** The player types **n** or **p** to swap cameras, looking for changes to item lists, and reporting those changes with **r** until the time limit, or until there are too many anomalies. An anomaly is any change to the room's items or description.

## New Features

For this assignment, you will be selecting from a small list of features that you will program and add to the game in Assignment 9 and writing out their requirements and a prioritized task list. You will be starting with the provided **RunGame.py** code and building from this, simulating more authentic programming environments.

Make sure to try playing the game by running **RunGame.py** before continuing. You and your partner must pick **one** new feature from **each** category, for a total of **two** features that need to be planned in this assignment and then implemented in Assignment 9. In all cases, plan to contain your features inside of simple functions, following good code design practices wherever possible.

### Category 1: Setup from Files

You must pick **one** setup feature to add. In both cases, the user will specify a **text file name** using a **command line argument** to change the setup of the game without needing to be a programmer. The file can be formatted in any way that is easy to work with in a text editor and in the Python program. **Use of the file should be optional,** and the program should work **even without a command line argument.**

1. **Loading Game Settings from a File:** The user will specify a file that contains settings to load the game with. Rather than setting settings directly in the Python file, all game settings will be loaded from the file letting users easily adjust their own difficulty settings without needing to program. The default settings in the **Duty** module's `init()` function are used if a command line argument is not provided.

2. **Add Rooms to a File:** The user will specify a file that contains **room names** and a **list of items for each room.** Rather than adding rooms directly in the Python file, all game rooms and items will be loaded from the file. The **filename** will be specified via a **command line argument,** allowing users to create their own maps without needing to program. The file can be formatted in any way that is easy to work with in a text editor and in the Python program. The default rooms provided in **RunGame.py** are used if a command line argument is not provided.

### Category 2: New Anomaly

You must pick **one** new anomaly to add into the game. Some may be more challenging than others. In all cases, if the anomaly **cannot be added to a room** (i.e., `add_anomaly()` returns **False**), it should try another.

1. **Number Anomaly:** Changes an item with a number in it so that it displays a different number that is one higher or lower at random. More items with numbers will need to be added for testing.
2. **Material Anomaly:** Replace words that are materials in an item to use a different material (e.g., Leather, Metal, Wooden, Plastic). Define your own materials to work with your own rooms.
3. **Colour Anomaly**: Replace words that are colours with other colours. Define your own list of colours that can be replaced to work with your own rooms.
4. **Typo Anomaly**: Pick a random letter in an item description and either add an extra one of those letters OR replace it with a similar looking character (e.g., **Sofa** could become **Soofa** or **$ofa**). Your program must be able to do **both** forms of typo.

At the end of this document, you will find the appendices which detail the game settings and the documentation for the **Duty.py** functions you might need to use during this assignment.

## Planning Worksheets

These worksheets have been provided as a separate, editable .docx file on Brightspace within the same Assignment module: **comp1405_f23_assignment_08.docx.** Download the .docx file and fill it out together with your partner. Save the .docx file as a PDF and submit a single .pdf file to the assignment drop box. **Both partners should submit a copy of the PDF to the assignment drop box individually.**

### Step 0: Features

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

1. **Category 1:** <Feature Selection>
2. **Category 2:** <Feature Selection>

### 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>**"

1. As a **<role>,** I want to **<action>,** so that I can **<motivation>**
2. As a **<role>,** I want to **<action>,** so that I can **<motivation>**

### 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 cannot be modified.**

#### Category 1

1. **Constraint:** Cannot modify the Duty.py module

#### Category 2

1. **Constraint:** Cannot 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.

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

1.

## 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.

**Category 2 Task List(s)**

1.

## 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:**<br><br>**1.**<br><br>**2.** |
| **What days are you planning to meet to work together to code Assignment 9, in-person or online?** |
| **How are you planning to pair-program together (e.g., screen sharing, together in-person, VS Code Live Share)?** |
| **How long do you each anticipate working on the code for this assignment?** |
| **Who will be submitting this PDF and who will be submitting the Assignment 9 Python file?** |

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.

## 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?
2. What functions from the **RunGame.py** file can you, and must you, modify or use as an example?
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?

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.

**Write your responses either here or on later pages.**

## Appendix A: Game Settings

The game can be configured to have different **settings**. These are values which modify the behaviour of the game. These do not need to be changed but can be to modify the difficulty among other things. These settings can be sent as **keyword arguments** to the **Duty** module's `init()` function OR set using the **Duty** module's `set_setting(setting: str, value)` function.

| Setting Keyword | Description | Default Values |
|---|---|---|
| **debug** | Prints helpful messages to help you debug your code or keep track of the game's state. | **False** |
| **timescale** | Number of in-game seconds which pass for every real-world second. | **10.0** (for 10x speed) |
| **probability** | Probability of an anomaly occurring each in-game minute | **0.1** (for 10% chance) |
| **max_anomalies** | Max number of active anomalies before losing (must be at most the number of rooms). | **4** (requires 4 rooms) |
| **anomaly_report_time** | Number of real-world seconds it takes to file a report to discourage false reporting. | **5.0** |
| **max_seconds** | In-game seconds to reach without too many anomalies active until winning the game. | **18000.0** (5 in-game hours) |
| **min_seconds_between_anomalies** | Minimum number of in-game seconds where anomalies will not occur one after another. | **1200.0** (20 in-game minutes) |

**Example: Settings using `Duty.set_setting(keyword, value)`:**

```python
Duty.set_setting("debug", False)
Duty.set_setting("timescale", 60)
Duty.set_setting("probability", 0.1)
Duty.set_setting("min_seconds_between_anomalies", 10*60)
```

## Appendix B: Game Functions

There are many functions available to you in the **Duty.py** module, and a few sample usages in the **RunGame.py** file that you can work from. **You are not expected nor encouraged to read Duty.py.**

- Functions you can use **exactly as used** in the provided `main()` function and **shouldn't need to learn**:
  - `init()`, `display()`, `handle_input()`, and `update()`:
    - Used in the correct order in the provided **RunGame.py**
    - `init()` will setup the game and print the initial message
    - `update()` updates the game state, returning True if the game should continue running
    - `display()` prints the game state to the player
    - `handle_input()` prompts for user input and runs the appropriate functions to handle it
  - `number_of_anomalies_to_create() -> int`:
    - Simulates  time passing between user inputs.Returns an integer number of how many anomalies you should attempt to create.

- Functions to **help set up the game** in the same way it's been provided in `main()`
  - `register_anomaly(name: str):` Adds the name of the anomaly to the list of valid anomalies so that it can be reported. Use any time you design a new anomaly and want it in game. Example usage in the `register_anomalies()` function in **RunGame.py**
  - `set_setting(setting: str, value):` Sets game settings using the settings keywords from Appendix A, to support feature expansion and adjusting game difficulty.
  - `add_room(room_name: str, room_items: list[str]):` Adds a new room to the game with the given name and list of items in the room without any anomalies. Example usage in the `add_rooms()` function in **RunGame.py**

- Functions to **help add anomalies to rooms**, with example usages in **RunGame.py**:
  - `get_random_anomaly() -> str:` Return the name of a random anomaly from the list of registered anomalies, used to determine which function to call to add an anomaly.
  - `add_anomaly(name: str, room, modified_item_list: list[str]) -> bool:` Tries to add the name of the anomaly into the room, which accepts either the name of the room or the index in the room list. It will set the item list to the `modified_item_list` parameter, and the `modified_item_list` should be the new, changed items. Returns `True` if it worked, `False` if it didn't, possibly because the modified items were the same. It will raise an error if the anomaly name was not registered with `register_anomaly().`
  - `get_rooms() -> list[str]:` Returns the names of all of the rooms in order.
  - `get_room_items(room) -> list[str]:` Returns the list of items in the room without any anomalies. Room can either be the name of the room or an integer. Can be used to modify the item listing or check if the items are valid for your new anomaly.
  - `rooms_without_anomalies() -> list[str]:` Returns the names of rooms in order that have no anomalies in them. Could be used to help pick a room to add an anomaly into.
  - `get_random_unchanged_room() -> str:` Return the name of a random room that does not have an anomaly currently in it. Used to pick a room to add an anomaly into.