|  |  |
| --- | --- |
| **Task:** | **AT2** |
| **Task Title:** | **Portfolio** |
| **Task Code:** | **AT2 POR Task 2** |
| **Name:** |  |
| **Student ID:** |  |

Table of Contents

[Scenario 2](#_Toc94868090)

[Core requirements 2](#_Toc94868091)

[Additional requirements 3](#_Toc94868092)

[The player’s Inventory 3](#_Toc94868093)

[File-based map 4](#_Toc94868094)

[General game play of text adventures 4](#_Toc94868095)

[Getting started 7](#_Toc94868096)

[Get feedback 7](#_Toc94868097)

[Submission of Portfolio Work 7](#_Toc94868098)

# Scenario

You are working at a boutique Perth-based software development company.

You will be writing a simple, single player, vintage text-based adventure game in Python3 (version 3.8 or higher).

The game will consist of different locations (rooms, etc.) which will be placed in a simple 2-dimensional grid (like a Chess board).

Not all locations in the grid need to be populated.

Players can only move between adjacent locations and each location must have at least one neighbour that can be reached. Players cannot move between rooms diagonally.

Locations do not have to be reachable from **all** neighbouring locations. For example, a room might have four neighbouring locations and be accessible from only two.

The program must be modular, so an object-oriented approach is recommended.

Follow all guidelines (see Assessment Task), including PEP 8.

### Core requirements

Your solution must have:

* A name, which you must choose. It should reflect the world of your adventure.
* A world, which you must design. Your world may be indoors, outdoors, or a combination of both.
* At least 10 locations (a location can be a space indoors or outdoors); a location must have a unique name, a description, and zero or more objects. The adventure always starts in a specific location.
* The ability to navigate the world. For example, you can indicate the directions a player may move to: N (North), S (South), W (West), and E (East). Your program must always indicate the available directions for the current location of the player.
* At least 3 characters (persons, animals, etc.), not including the player; each character must interact in some way with the player. They should have unique names.
* The ability to interact with the characters, like talking or asking questions.
* At least 10 objects that can be fixed (cannot be picked up) or not fixed (can be picked up). Objects may also be invisible/visible, which may depend on another condition (example provided below). Some (non-static) objects must be able to be picked up by the player and there should be at least 5 of those. The player should be able to “look at” objects to get additional information.
* The ability to look at and pick up items and put them away for future use.
* The player must have a “bag” or a “rucksack” for storing objects while moving around. See additional instructions below in the section “The player’s inventory”.
* At least one conditional action, for example, a door that can only be opened if the player has a key. When the player tries to open the door without having possession of the key, the program should say, for example, “The door is locked. How about you find a key first?”
* An option to store a simple representation of the “map” in a file; this file must be updated using random-access techniques. See additional instructions below in the section “File-based map”.
* A goal. In other words, the player should be able to finish the game by fulfilling a certain task. Additionally, the player may “die” if they make a wrong move. (For example: “You pressed the big, red button that says ‘DON’T PUSH’. A bomb exploded. You are dead.”)

### Additional requirements

Your program:

* Should be developed in an integrated development environment (IDE), preferably PyCharm.
* Must be modular; using an object-oriented approach is encouraged. You may apply any of the UML techniques which you have used in Portfolio Assessment AT2.
* Must be tested in a meaningful way; unit testing is encouraged.
* Must be documented, i.e., use docstrings and other internal documentation techniques. Provide clear instructions to the player either within the program or in a separate document.

### The player’s Inventory

The player can hold multiple objects at the same time. This is called the “inventory” and can be a “bag” or a “rucksack” from the player’s perspective.

You will be given an (unfinished) class that will provide the functionality of the “bag”. It will store objects using their (unique) name in a list. Finding objects must be done using binary search.

Follow the instructions in the source file to complete this class and use it in your program.

### File-based map

You are required to keep track of the locations that have been visited by the player in a text file. For this purpose, you must use random access technique. You will be given an (unfinished) class that will provide this functionality.

Follow the instructions in the source file to complete this class and use it in your program.

The map will simply indicate whether a location has been visited with an **X**. An unvisited space is indicated with a ▢ (space).

### General game play of text adventures

Vintage text adventures follow a certain pattern that defines their game play. For example, when the player starts the game, they will be given a description of the (creepy?) environment and some options:

You are standing at the North end of the forest. It is cold and the environment feels hostile. You probably don’t want to stick around longer than necessary. Roads leads South and East.

What do you want to do? Exits: S, E

In any case, all available “exits” (places to go from the current location) are explicitly shown on the screen.

#### Interaction

Text adventures become more fun as the player interacts with characters and objects. For example, characters may give the player a hint, but perhaps only if you give them something of value:

You are in the middle of the forest. A gnome is watching you suspiciously. He looks like he knows something and seems corruptible.

What do you want to do? Exits: N, S, E, W

* Ask information

“Happy to help, Stranger. What is this information worth to you?” The gnome senses you may have a dime to spare.

What do you want to do? Exits: N, S, E, W

* Give dime

“Thanks, Stranger. Find John, he holds the key to your puzzle.” The gnome vanishes.

This way, the player will have to solve puzzles created by you to finish the game. The “key” in this case can be a literal key with which you can later open a door. Now, you’ll just have to find John.

#### Commands

Commands in text adventures are either single characters (for example, just type N to go North) or a combination of two words: <verb> <noun>, like “get coin” or “look clock”. To keep things manageable, you may follow this convention.

For example, you might find an object that you would like to investigate:

You have entered the library. There are many books along the walls and a few clocks. The grandfather clock literally stands out.

What do you want to do? Exits: S, E, W

* Look clock

The grandfather clock is standing tall. It has stopped running a long time ago at 3:27. Still, it shows the correct time twice a day.

What do you want to do? Exits: S, E, W

You can follow up with additional questions, which may or may not be useful. Maybe the time that the grandfather clock shows is a hint? A code?

#### Objects

You can have different types of objects. For example:

* *Fixed*: static objects are fixed and cannot be picked up (like the grandfather clock from the previous example). The player may still be able to interact with it in some way.
* *Not fixed*: this type of object can be picked up and be moved. For example, a coin that you find somewhere on the Road to Nowhere. You can pick up objects using “get <object>”. Objects that are picked up should be stored in the player’s “rucksack” and stay with the player as they move through the world.
* *Invisible*: an object that is at a location but is not visible (and can therefore not be picked up). Such objects may become visible after another action has taken place (in other words: they have a precondition that must be satisfied for the object to become visible). Here is an example:

You are at an open patch in the forest. There is a small heap of leaves just in front of you.

What do you want to do? Exits: S

* Blow leaves

You have blown away the leaves and a shiny gold coin has appeared.

What do you want to do? Exits: S

* Get coin

You have put the coin in your rucksack.

What do you want to do? Exits: S

Using the command Inventory (or I for short), you can get a list of all items in your possession:

* I

You are holding: a gold coin.

What do you want to do? Exits: W, E

The program should guide the player to quite some extent. That is why you should repeat the question “What do you want to do?” and show the available exits after each move played. This will make it super clear to the player that they have to perform an action.

You are encouraged to build in little jokes. For example, in the 1980’s text adventure “Sherwood Forest” the player can talk to an owl, which then responds with “Who?” Hilarious!

Your program, your world, and everything in it, must be “suitable for work” (SFW).

Do NOT put in any rude jokes or puns.

### Getting started

You should think about the world you want to build before writing a single line of code. Use paper and a pen or pencil to make a rough sketch of your world’s layout.

Think about the goal you want the player to achieve. Then, create the various puzzles that the player must solve, working your way backwards to the start of the game (from the player’s perspective).

Don’t go overboard with these puzzles. It’s better to have two or three well thought out puzzles than to have a dozen lame ones. Also, it should not take hours to complete the game (in this case).

Feel free to use existing stories to base your world on. For example, if you are a Harry Potter fan, you can base your world on Harry Potter. That will probably give you enough inspiration to come up with locations, characters, etc.

Try to be consistent and prevent the player from having to guess. Provide as many hints as you can without making it too obvious what the player needs to do. Hints can come from multiple sources: the description of a location (example: the leaves from earlier), the description of an object, or a character (remember the gnome?).

Make sure that everything has a function in your world. For example, you should not have objects that can be picked up that serve no purpose.

Use the UML modelling techniques you have learned to your advantage. It is not mandatory to use UML modelling for this assessment, though.

You must do all your coding in a Git repository with a remote in GitHub. That way, you can keep track of your progress. It also acts as a backup of your code and can help you recover from mistakes. You can also do lots of experiments in different branches without breaking the main code.

### Get feedback

Get feedback early and often! While writing a text adventure in Python is totally doable, it is not completely trivial. Ask for help as much and as often as you want. Your lecturers will provide you with guidance without giving you the answers (obviously).

### Submission of Portfolio Work

To submit the portfolio, do the following:

* Ensure you have put your name and student ID on the front page of this document. Your submission will not be accepted if name or student ID is missing.
* Save any Word documents with the correct extension (.docx).
* Do NOT zip your Word document!
* Create a zip file with your Python project (please remove the virtual environment and project folder from the zip file before submitting).
* Open Blackboard, and locate the AT3 PRJ Project Assessment
* Open the assessment and upload the original documents and the zip file as separate files in one submission.
* Click submit.