**Assessment type (🗹):**

Questioning (Oral/Written)

Practical Demonstration

3rd Party Report

Other – Project

**Assessment Resources:**

|  |
| --- |
| * Access to PC with Python3, a Python IDE (e.g., PyCharm) * Access to Word or Office 365 * Access to Internet * Access to Blackboard (for submission) * Policies and procedures required for this assessment * Project Scenario (separate document) |

**Assessment Instructions:**

|  |
| --- |
| 1. To complete this assessment, you must follow the steps in the Assessment Instrument. 2. All questions and tasks in this assessment must be addressed fully to meet the assessment requirements. 3. All required skills and knowledge must be demonstrated to achieve a satisfactory result. 4. Include references if you use any external resources. You do **not** have to use APA referencing. 5. The duration of this assessment is 6-8 hours, which can be spaced over multiple weeks. 6. The assessment is preferably done during class hours. This will ensure authenticity and it will allow the student to ask questions to the assessor/lecturer. 7. Make sure your code follows the guidelines provided in the Project Scenario. This includes but is not limited to the proper use of comments and docstrings as well as formatting. You must follow PEP 8 for any Python code. 8. You must use a repository (on GitHub) to keep track of your coding progress. Give your lecturer access once you have created the repository. 9. You must upload your code (preferably a PyCharm project) as a zip file to Blackboard. Upload any Word document as separate files in the same submission. Do not zip the Word document and do not put the Word document in the Python project folder. 10. This assessment is due in Week 18 (the actual date can be found on Blackboard). |

**Assessment Instrument:**

|  |
| --- |
| Download and read the Project scenario carefully and follow the instructions.  Additional steps are as follows (provide evidence):   * You must follow the user requirements in the project scenario (core and additional). * You must define aggregate data structures and use them in your program. * Your program must consist of multiple source files. * You must use debugging tools (standalone and provided by the IDE); trace code and examine variables in memory. * Apply consistent and useful internal documentation in the form of comments (single line and docstrings). * You must test your program against the requirements and capture and document the results of the tests. * Follow PEP 8 The Style Guide for Python throughout the entire process. * Follow any other policies and procedures that are provided in the context of this assessment.   Your assessor will use the following checklist to mark your work. A more verbose description can be found in the Project Scenario, which is the leading document for the adventure’s requirements. |

| **Checklist:** | |
| --- | --- |
|  | The program has a name that reflects the adventure’s world. |
|  | The program has a world that contains at least 10 locations. Each location has a unique name, a description, and zero or more objects. |
|  | The program allows the player to navigate the world. It should at least offer to go in the directions N, E, W, and S. The program shows available exits from every location. The player can move in these four directions by only typing the corresponding letter. E.g., entering just the letter N will mean the player moves North. |
|  | The adventure has at least three additional characters (excluding the player). Each character must have a unique name. The characters can be people, animals, fictional characters, etc. |
|  | The player can interact with the additional characters (talk, ask questions, etc.) in a meaningful way. |
|  | The adventure should have at least ten different objects. Each object must be uniquely named. Objects can be static or not static, visible or invisible, etc. Some (non-static) objects must be able to be picked up by the player and there should be at least five of those. The player should be able to “look at” objects to get additional information. |
|  | For the purpose of holding objects, the player should have a “rucksack” or similar holding device (i.e., an Inventory). |
|  | The inventory may hold multiple objects at a time. Objects should be stored using their unique names. Objects must be searched using a simple binary search technique. |
|  | The adventure must have at least one conditional action. A conditional action is an action that can only be performed if a certain prerequisite holds. For example, a certain door may only open if the player has a key. |
|  | The adventure must store a simple map (2D) in a file. This map should be updated using a random-access algorithm. The map should have X’s for places that have been visited and a ▢ (space) for places that have not been visited. |
|  | The adventure must have a certain goal: that is how to win the game. For example: “Your task is to re-engage the safety system of the reactor core to prevent a core meltdown.” In addition, it’s possible for the player to lose the game (“game over”) under certain conditions.  After winning or losing the game, the players should be able to start a new game. |
|  | The adventure was developed using an IDE. |
|  | The adventure must be built using a modular approach, preferable in an object-oriented fashion. This means multiple sources files must be used, for example, one per class. |
|  | The adventure must be tested in a useful way. Unit tests are encouraged, but a system test is acceptable too. |
|  | The code must have useful documentation, e.g., docstrings, comments, etc. |
|  | The adventure must have clear instructions for the player. No guesswork as to how to run the program is allowed. The instructions must also explain how to win the game. |
|  | The adventure should follow the “General game play of text adventures” that is described in the Project Scenario. This includes:   * A starting description of the game * Show available exits (mentioned before) * Interaction with characters, e.g., “ask information”, “give dime”, etc. * Use commands like “look clock”, “get key”, etc. Commands should be kept simple, e.g., in the form of “<verb> <noun>”, for example, “light match”. * Objects: fixed, not fixed, (in)visible, having a condition * List player’s possessions with the I or Inventory command |
|  | The adventure MUST NOT have any unsuitable jokes in it, e.g. no rude jokes or puns. [Easter eggs](https://en.wikipedia.org/wiki/Easter_egg_(media)) are allowed, though, as long as they are SFW. |
|  | Student has requested feedback early and often during the assessment process. |
|  | The solution has been committed to a GitHub repository and a history of commits is readily available. |
|  | The code is written in Python3, version 3.8 or higher. |
|  | The student followed all guidelines, including PEP 8 for writing Python code. |
|  | The submission’s front page has student name and ID. |