IT 140 Project Two Guidelines and Rubric

Competencies

In this project, you will demonstrate your mastery of the following competencies:

- Write scripts using syntax and conventions in accordance with industry standard best practices
- Develop a fully functional program using industry-relevant tools

Scenario

You work for a small company that creates text-based games. You recently pitched your design ideas for a text-based adventure game to your team. Your team was impressed by all of your designs, and would like you to develop the game! You will be able to use the map and the pseudocode or flowcharts from your designs to help you develop the code for the game. In your code, you have been asked to include clear naming conventions for functions, variables, and so on, along with in-line comments. Not only will these help you keep track as you develop, but they will help your team read and understand your code. This will make it easier to adapt for other games in the future.

Recall that the game requires players to type in a command line prompt to move through the different rooms and get items from each room. The goal of the game is for the player to get all of the items before encountering the room that contains the villain. Each step of the game will require a text output to let the player know where they are in the game, and an option of whether or not to obtain the item in each room.

Directions

In Project One, you designed pseudocode or flowcharts for the two main actions in the game: moving between rooms and gathering items. In this project, you will write the code for the full game based on your designs. You will also need to include some additional components beyond your original designs to help your game work as intended. You will develop all of your code in one Python (PY) file, titled "TextBasedGame.py."

IMPORTANT: The directions include *sample* code from the dragon-themed game. Be sure to modify any sample code so that it fits the theme of *your* game.

1. First, create a new file in the PyCharm integrated development environment (IDE), title it "TextBasedGame.py," and include a comment at the top with your full name. As you develop your code, remember that you must **use industry standard best practices including in-line comments and appropriate naming conventions to enhance the readability and maintainability of the code**.
2. In order for a player to navigate your game, you will need to **develop a function or functions using Python script**. Your function or functions should do the following:
    - Show the player the different commands they can enter (such as "go North", "go West", and "get [item Name]").
    - Show the player's status by identifying the room they are currently in, showing a list of their inventory of items, and displaying the item in their current room.

    You could make these separate functions or part of a single function, depending on how you prefer to organize your code.

```
#Sample function showing the goal of the game and move commands
def show_instructions():
   #print a main menu and the commands
   print("Dragon Text Adventure Game")
   print("Collect 6 items to win the game, or be eaten by the dragon.")
   print("Move commands: go South, go North, go East, go West")
   print("Add to Inventory: get 'item name'")

#In this solution, the player’s status would be shown in a separate function.
#You may organize your functions differently.
```

3. Next, begin developing a main function in your code. The main function will contain the overall gameplay functionality. Review the Project Two Sample Text Game Flowchart, located in the Supporting Materials section, to help you visualize how main() will work.
   For this step, simply add in a line of code to define your main function, and a line at the end of your code that will run main(). You will develop each of the pieces for main() in Steps #4–7.

4. In main(), **create a dictionary linking rooms to one another and linking items to their corresponding rooms**. The game needs to store all of the possible moves per room and the item in each room in order to properly validate player commands (input). This will allow the player only to move between rooms that are linked or retrieve the correct item from a room. Use your storyboard and map from Project One to help you create your dictionary.
   Here is an example of a dictionary for a few of the rooms from the sample dragon text game.

```
#A dictionary linking a room to other rooms
#and linking one item for each room except the Start room (Great Hall) and the room containing the villain
rooms = {
   'Great Hall' : { 'South' : 'Bedroom', 'North': 'Dungeon', 'East' : 'Kitchen', 'West' : 'Library' },
   'Bedroom' : { 'North' : 'Great Hall', 'East' : 'Cellar', 'item' : 'Armor' },
   'Cellar' : { 'West' : 'Bedroom', 'item' : 'Helmet' },
   'Dining Room' : { 'South' : 'Kitchen', 'item' : 'Dragon' } #villain
}
#The same pattern would be used for the remaining rooms on the map.
```

5. The bulk of the main function should include a **loop for the gameplay**. In your gameplay loop, **develop calls to the function(s) that show the player’s status and possible commands**. You developed these in Step #2. When called, the function(s) should display the player’s current room and prompt the player for input (their next command). The player should enter a

command to either move between rooms or to get an item, if one exists, from a room.

Here is a *sample* status from the dragon text game:

```
You are in the Dungeon

Inventory: []

You see a Sword

----------------------

Enter your move:
```

As the player collects items and moves between rooms, the status function should update accordingly. Here is another example after a player has collected items from two different rooms:

```
You are in the Gallery

Inventory: ['Sword', 'Shield']

--------------

Enter your move:
```

Note: If you completed the Module Six milestone, you have already developed the basic structure of the gameplay loop, though you may not have included functions. Review any feedback from your instructor, copy your code into your "TextBasedGame.py" file, make any necessary adjustments, and finish developing the code for the gameplay loop.

6. Within the **gameplay loop**, you should include **decision branching to handle different commands and control the program flow**. This should tell the game what to do for each of the possible commands (inputs) from the player. Use your pseudocode or flowcharts from Project One to help you write this code.
    - What should happen if the player enters a command to move between rooms?
    - What should happen if the player enters a valid command to get an item from the room?

   Be sure to also include **input validation** by developing code that tells the program what to do if the player enters an invalid command.

   Note: If you completed the Module Six milestone, you have already developed a portion of this code by handling "move" commands. Review any feedback from your instructor, copy your code into your "TextBasedGame.py" file, make any necessary adjustments, and finish developing the code.
7. The **gameplay loop should continue looping, allowing the player to move to different rooms and acquire items until the player has either won or lost the game**. Remember that the player wins the game by retrieving *all* of the items before encountering the room with the villain. The player loses the game by moving to the room with the villain *before* collecting all of the items. Be sure to include output to the player for both possible scenarios: winning and losing the game.

Hint: What is the number of items the player needs to collect? How could you use this number to signal to the game that the player has won?

Here is a sample from the dragon text game of the output that will result if the player wins the game:

```
Congratulations! You have collected all items and defeated the dragon!

Thanks for playing the game. Hope you enjoyed it.
```

If the player loses the game, they will see the following output:

```
NOM NOM...GAME OVER!

Thanks for playing the game. Hope you enjoyed it.
```

Note: If you completed the Module Six milestone, the gameplay loop ended through the use of an “exit” room. You will need to remove the “exit” room condition and adjust the code so that the game ends when the player either wins or loses, as described above.

8. As you develop, you should be sure to **debug your code to minimize errors and enhance functionality**. After you have developed all of your code, be sure to run the code and use the map you designed to navigate through the rooms, testing to make sure that the game is working correctly. Be sure to test different scenarios such as the following:
    - What happens if the player enters a valid direction? Does the game move them to the correct room?
    - When the player gets an item from a room, is the item added to their inventory?
    - What happens if the player enters an invalid direction or item command? Does the game provide the correct output?
    - What happens if the player wins the game? What happens if the player loses the game?

What to Submit

To complete this project, you must submit the following:

TextBasedGame.py

Develop and submit the “TextBasedGame.py” file using PyCharm. Include your full name in a comment at the top of the code. Be sure to submit the code that you have completed, even if you did not finish the full game.

Supporting Materials

The following resources may help support your work on the project:

Video: Sample Dragon Text Game Walkthrough (8:24)

This video shows a sample dragon-themed text game. There is a brief description of the game, as well as a video that shows the game running and a player moving through different rooms and gathering items based on the commands. Review this video to help you understand how a text-based adventure game works. A video transcript is available: Transcript for Sample Dragon Text Game Walkthrough

Reading: Project Two Sample Text Game Flowchart

This flowchart outlines a sample design for the whole text-based game. The “Get Item” and “Move Between Rooms” processes are intentionally vague. You designed more detailed flowcharts or pseudocode for these processes as a part of your work on Project One. Use this flowchart as a guide when developing your code.

Reading: Sample Dragon Text Game Output

This document shows the sample inputs and outputs for the dragon-themed text game. Review the sample inputs and outputs to better understand how a text-based adventure game works.

Reading: Getting Started With PyCharm

This document walks you through a step-by-step process for downloading PyCharm to get started with your first PyCharm project. You will also need to download Python in order for PyCharm to run, which is detailed in its own section of this tutorial.

Video: Pycharm Tutorial #1 - Setup and Basics (12:21)

This tutorial will orient you to the integrated development environment (IDE) PyCharm, where you will be writing and testing code for labs and projects. The tutorial will walk you through how to access and navigate PyCharm, how to run code, and how to write code. After watching the tutorial, practice logging in and creating files. If you have any questions, please reach out to your instructor. A video transcript is available: Transcript for Pycharm Tutorial #1 - Setup and Basics

Project Two Rubric

Criteria	Exemplary	Proficient	Needs Improvement	Not Evident	Value
Functions	N/A	Develops function(s) using Python script to organize code and meet required functionality (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include meeting all required functionality (55%)	Does not attempt criterion (0%)	10
Main Function: Dictionary	N/A	Creates a dictionary that meets required functionality (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include dictionary syntax or ability to meet required functionality (55%)	Does not attempt criterion (0%)	10

Criteria	Exemplary	Proficient	Needs Improvement	Not Evident	Value
Main Function: Gameplay Loop	N/A	Applies a loop to meet required functionality and control program flow (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include loop syntax, logic, or ability to meet required functionality (55%)	Does not attempt criterion (0%)	15
Main Function: Function Calls	N/A	Develops function calls to meet required functionality (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include proper syntax or application of function calls to meet required functionality (55%)	Does not attempt criterion (0%)	5
Main Function: Decision Branching	N/A	Applies decision branching to meet required functionality and control program flow (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include decision branching syntax, logic, or ability to meet required functionality (55%)	Does not attempt criterion (0%)	20
Input Validation	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or efficient manner (100%)	Validates all instances of user input (85%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include validating all instances of user input (55%)	Does not attempt criterion (0%)	20
Debugging	N/A	Debugs code to minimize errors and enhance functionality (100%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include making sure all possible inputs and tests work for the program (55%)	Does not attempt criterion (0%)	10

Criteria	Exemplary	Proficient	Needs Improvement	Not Evident	Value
Industry Standard Best Practices	Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner (100%)	Uses industry standard best practices including in-line comments and appropriate naming conventions to enhance readability and maintainability of code (85%)	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include detail of in-line comments or appropriateness of naming conventions (55%)	Does not attempt criterion (0%)	10
				Total:	100%