**CIS 162 Project 3 (Part B)**

**Quest Game with NPC**

**Before Starting the Project**

* This is a challenging project! It is important that you read this entire project description before starting and to get started right away.

**Learning Objectives**

After completing this project you should be able to:

* *use* a List to maintain and process collections of objects (zyBook Ch 8)
* *use* a Dictionary to maintain and process collections of objects (zyBook Ch 8)
* *develop* an application with multiple classes

**Step 1: Create a New PyCharm Project**

**Step 2: Create a python file called item.py (5 pts)**

Within the file, write a class called Item to maintain information about an item including: a one-word name (String), a longer description of the item (String), and a weight (int). It is good practice to provide set and get methods for most instance variables.

**Instance Variables**

Identify appropriate names and data types for each of the three instance variables.

**Methods**

* \_\_init\_\_(self, name, desc, weight) – assign provided values to the instance variables.
* \_\_str\_\_(self)– return the item description. Refer to section 9.8 about this special method. DO NOT print anything in this method! Instead, just return the description.
* get\_weight(self) – return the item weight.
* get\_name(self) – return the item name.
* get\_description(self) – return the item description.
* set\_weight(self, wt) – update the item weight.
* set\_name(self, name) – update the item name.
* set\_description(self, desc) – update the item description.

Test this class thoroughly on your own before moving on to the next class.

**Step 3: Create a python file called npc.py (5 pts)**

Within the file, write a class called NPC to maintain information about a non-player character (NPC) including: a name (String) and a spoken phrase (String). It is good practice to provide set and get methods for most instance variables.

**Instance Variables**

Identify appropriate names and data types for each of the instance variables.

**Methods**

* \_\_init\_\_(self, name, phrase) – assign provided values to the instance variables.
* get\_phrase(self)– return the spoken phrase.
* get\_name(self)– return the name.
* set\_phrase(self, phrase)– update the spoken phrase.
* set\_name(self, name)– update the item name.
* speak(self)– return a formatted String that includes the name and spoken phrase. For example if the name is “Louie the Laker” and their phrase is “Go Lakers!”, speak() would return the following string.

Louie the Laker says, “Go Lakers!”

**Alert!** Duplicate the format using name and phrase, including punctuation and double quotes around the phrase to pass the automated tests.

Test this class thoroughly on your own before moving on to the next class.

**Step 4: Create a python file called room.py (10 pts)**

Within the file, write a class called Room to maintain information about a room. Rooms can have an unlimited number of neighbors in unique directions. For example, a Room could have neighbors to the “north”, “upstairs”, “outside”, “southwest” or any other direction. Allowing an unknown number of neighbors requires a dynamic data structure like a Dictionary.

**Note:** the class should have no references to specific rooms in your game. It can be used for any game design without any changes.

**Import**

You will need to import the Item and NPC classes at the start of this file, before the class.

from item import Item

from npc import NPC

**Instance Variables**

Identify meaningful names with appropriate data types for each of these instance variables:

* a description of the current location (String)
* an optional item (Item)
* an optional character (NPC)
* a list of all adjacent rooms (Dictionary)
  + direction is the key, room is the value

(These are the instance variables for your constructor)

**Methods**

* \_\_init\_\_(self, desc, thing = None, character = None) – a constructor that is passed the description, an item (optional), and a NPC (optional).
* get\_item(self) – return the item.
* get\_npc(self) – return the NPC.
* get\_description (self) – return the location description.
* set\_item(self, thing) – add an item to the location. Rooms only hold one item at a time. If an item already exists, it is replaced by the new item.
* set\_npc(self, npc) – add an NPC to the room. Rooms only hold one NPC at a time. If an NPC already exists, it is replaced by the new NPC.
* set\_description (self, desc) – replace the room description.
* has\_item(self) – return True if the location has an item. Otherwise, False.
* has\_npc(self) – return True if the Room has a NPC. Otherwise, False.
* add\_neighbor(self, dir, rm) – add the provided Room and corresponding direction to the Dictionary of neighbors. This is only one line of code! Refer to Ch 8. Modify the following example as needed:

self.neighbors[dir] = rm

* get\_neighbor(self, dir) – return the neighboring Room in the requested direction. Confirm there is a neighbor in the requested direction before returning. Otherwise, return None.
* remove\_item(self) – remove the item from the Room and return it. As a result, the instance variable for the item is set to None. Warning: this can be a bit tricky!
* \_\_str\_\_(self)– return a String that begins with “You are” followed by the location description. If there is an item, include “You see” followed by the item description. If there is a NPC, include “You meet” followed by the NPC name. Refer to section 9.8 about the special \_\_str\_\_() function. DO NOT print anything here. Here is a sample:

You are in the kitchen.

You see a jar of peanut butter.

You meet Louie the Laker.

* Test this class thoroughly on your own before moving on to the next class.

**Preliminary zyBook Testing**

Perform a preliminary test by uploading “item.py”, “npc.py”, “room.py”, and an empty “game.py” to zyBook section 18.3 (you can add game.py to your project without any code – that is the next step). Note, you will not copy and paste code as you have in the past. The Submit for Grading button will not appear until all files are provided. Your solution should pass the first four tests if all functions have the correct name and perform correctly.

A screenshot of a computer

Description automatically generated

**Figure 1. zyBook interface to upload python files for testing in section 18.3.**

**Step 5: Create a python file called game.py (55 pts)**

Within the file, create a class called Game. This is the most complex class you have written in the course so far! It is responsible for keeping track of the game environment, player’s items, player’s current location and oh so much more.

**Import**

Import the Item, NPC and Room classes at the start of this class.

from item import Item

from npc import NPC

from room import Room

**Instance Variables (constructor)**

Use meaningful names with appropriate data types for each of these instance fields:

* multiple Room, Item and NPC variables are defined in the game (e.g. Kitchen, Basement, Candy, Elmo the Elf)
* a List of Items the player is currently holding. These items have been taken along the way.
* a Room to store the player’s current location.
* a String to maintain the current game message. Updated in almost every method.

**Constructor**

A *constructor* initializes variables to appropriate starting values.

* \_\_init\_\_( self) – performs four tasks to create the game environment. 1) Instantiate the List of Items. The list starts empty unless the player begins the game with items. 2) create all items, NPCs, and rooms by invoking create\_world() defined below. 3) Set the current room to the starting location of the game. 4) Set the initial message by invoking self.set\_welcome\_message() defined below.
* Also needs a message variable initiated to an empty string. This message is updated in almost every method.

**First Draft Methods (see early due date)**

* get\_message(self) – return the game’s message using one line of code. DO NOT print the message.
* get\_current\_room(self) – return the game’s current location using one line of code.
* create\_world(self) – create all items, NPCs, rooms, and neighbors. It is critical to perform these steps in the correct order. 1) Create all items first. 2) Create all NPCs. 3) And then create all rooms. 4) And finally connect all neighbors. For example:

self.shoe = Item("shoe", "a red Nike shoe", 10)

self.louie = NPC("Louie the Laker", "Go Lakers!")

self.outside = Room("outside the entrance", self.shoe)

self.theater = Room("in a lecture theater", None, self.louie)

self.outside.add\_neighbor("east", self.theater)

* move(self, dir) – if appropriate, update the current location with the neighbor in the requested direction. If not possible, the message should explain the player cannot move in that direction. Moving from one room to another involves updating the current location. Alter the following example as needed to work in your code:

next = self.current.get\_neighbor(dir)

if (next == None):

self.msg = "You can't move in that direction"

else:

self.current = next

self.msg = self.current.\_\_str\_\_()

* set\_welcome\_message(self) – set the game’s message with a description of the background and goal of the game. Establish the tone of the game. DO NOT print the message.

**First Draft zyBook Testing**

Perform automated testing by uploading all files to zyBook. Your solution should pass the first five tests.

**Support Interactive Play**

All classes and methods come together to allow someone to play the game in a terminal window by typing commands and reading responses. Continue by adding the following methods to the Game class.

* parse\_command(self) – prompt the player for a command and return two strings, with the second one optional. Feel free to use this method as provided.

def parse\_command(self):  
 words = input("Enter>>> ").split()  
 first = words[0]  
 if len(words) > 1:  
 second = words[1]  
 else:  
 second = None  
 return first, second

* play(self) – This function will be lengthy and drives the entire game. It is the only function in the entire program that has print statements. This is a design principle that isolates all output to one portion of code. A programmer can easily redirect the output to a graphical user interface instead of a terminal window.

Print the initial game message and then use a loop that prompts the player to enter a command. The loop continues until 1) the game is over or 2) the player types ‘quit’. A series of if statements call each game function. Prompt the player for a command by calling parse\_command().

# print initial welcome message

print(self.get\_message())

# Loop until game over

first, second = self.parse\_command()  
if first == "move":  
 self.move(second)  
elif first == "look":  
 self.look()

print(self.get\_message())

**Player Actions**

Continue by adding the following methods to the Game class.

* look(self) – update the game’s message with the current room’s description. Modify this example using your variable names.

self.message = self.current.\_\_str\_\_()

* help(self) – update the game’s message with hints, suggestions and reminders about the game objective. Perhaps provide all valid commands and directions.

You are lost and alone. Find three magic gems and shout the

magic phrase to escape.

* items(self) – update the game’s message with 1) a list of all items the player is holding or 2) a message indicating the player is not holding anything.

You are holding:

a red Nike shoe

a dusty old book

You are not holding any items

* take(self) – if appropriate, remove the item from the current location and add it to the List of items. Update the game’s message with one of the following options: 1) there is no item here to take, 2) the item is too heavy (50 units), or 3) the player is now holding the item.

There is nothing to take.

The treasure is too heavy to pick up!

You are holding a red Nike shoe

* place(self, name) – if appropriate, remove the item from the List of held items and add it to the current location. Update the game’s message with one of the following options: 1) the player is not holding that item, 2) the room already has an item, or 3) the player has successfully placed the item in the room. Details and format of each message is at your discretion.

You are not holding a magic wand.

There is already an item in the room.

You carefully place a diamond ring in the room.

* search\_items(self, name) – check the List of held Items by name. If found, return the Item. If not found, return None.

**Design Requirement** – Use a loop to step through the list.

* speak(self) – Update the game’s message with one of the following options: 1) the room has no NPC, 2) the NPC’s spoken phrase.

There is no one here to speak.

Frodo says, “Seek and you will find.”

* game\_over(self) – determine if the game is over because the player has either won or lost. If so, update the game’s message with the news and return True. Otherwise, return False and do not change the message.
* retreat(self) – the player attempts to return to the prior location. The player can only retreat one step before moving forward again. This action can be done even if there is no valid direction to the prior location. Also, it is not possible to retreat from the starting location either.

Update the game’s message with one of the following options: 1) cannot retreat from here, 2) the updated location description.

You cannot retreat from here.

You retreated.

You are in the misty cavern.

**Design Requirement** – add an instance variable to keep track of the prior location. The variable will be None when the action is not valid. Update several methods as needed.

* Add a function to implement one of the additional actions included in your game.
* Add a function to implement the second additional action

**Support Software Testing**

Provide the following method to demonstrate the game can be won. Show the necessary sequence of actions to win and the updated message after each action. Here is a short example that allows the player to win a simple game. The final message displayed should indicate the player won the game.

def auto\_win(self):

print(self.get\_message())

self.take()

print(self.get\_message())

self.move("south")

print(self.get\_message())

self.place("book")

print(self.get\_message())

self.move("north")

print(self.get\_message())

self.move("south")

print(self.get\_message())

self.take()

print(self.get\_message())

if(self.game\_over()):

print(self.get\_message())

**Support Instructor Testing**

Include the following code at the bottom of your Game class. This will help your instructor quickly test these methods.

if \_\_name\_\_ == '\_\_main\_\_':

*g = Game()*

*g.play()*

**Final zyBook Testing**

Perform all tests by uploading files to zyBook. Your solution should pass all tests at this point. However, this does not necessarily mean everything works correctly for your game if perfect. You need to do additional testing on your own.

# Coding Style (10 pts)

Good programming practice includes writing elegant source code for the human reader. Follow the [Style Guide for Python Code](https://peps.python.org/pep-0008/). For CIS 162, follow these additional guidelines:

* Comments are at the same indentation as the corresponding code
* Provide comments before most if statements and loops
* Blank lines before every comment
* Use docstrings to describe all classes and functions (see 5.9)

# Turn In

* Submit to Blackboard:
* Cover page - Provide a cover page that includes your name, a title, and an appropriate picture.
* Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the [School of CIS Guidelines for Academic Honesty](http://www.cis.gvsu.edu/academic-honesty/).
* Programming Journal (2nd page) – Professional coders track their time and journal their activities. What challenges did they encounter and how did they solve the issue? In addition, provide names of any people you helped or received help from.
* Source code - your elegant source code copied from PyCharm (with your name).

# Grading Criteria

* Part A Game Design (10 pts)
* Pass all zyBook tests (65 pts)
* Pass instructor tests during demo and solution meets all design requirements (10 pts)
* Signed pledge and programming journal (5 pts)
* Attractive source code with appropriate comments to describe separate sections of the solution. All files should have the following header. (10 pts)

*'''  
YOUR NAME  
Brief description of the code for other programmers  
  
I certify that this code is mine, and mine alone, in accordance with GVSU academic honesty policy.*

*COMPLETION DATE  
'''*