Assignment 3

Due Date: This assignment is due at 11:59:59PM on 5 November, 2013.

Submission: Solutions should be uploaded in a single-file attachment to the appropriate Assignment on `http://newclasses.nyu.edu`

Style: This assignment will be graded, in part, by automated testing, so all classes, members, methods, etc. must adhere **precisely** to that specified. Code should be well-commented, logically packaged, and easy to read. Variable names should be self-documenting wherever possible. No line should exceed 80 characters.

This assignment will require the use of stacks, queues, graphs, and overriding of equality. If you are lost, make liberal use of the books, online resources, and office hours. The maximum grade, including extra credit, is **120 points**. This Assignment will be worth the same, proportionally, as Assignment 1.

You will be building a simple dungeon game by wiring together rooms in a graph-like structure. Gameplay will be driven using a `main()` method and `raw_input`.

At a high level, you will be creating a `Room` class to represent the rooms; these will be interconnected to the north, south, east, and west as appropriate. Some of these interconnections between rooms will be blocked by gates that can be opened by using gems of particular colors; these latter will be represented by a class `GateGem`. Finally, at least one room will contain an exit.

Please **read carefully** and **make sure you test each class and method independently, and playtest the game**. Also take care that when submitted, your script does not contain any extraneous code. **No code may make use of globals other than imports.** Arguments **must** be passed in function calls. All output in function specifications is via `return` statements, all exceptions are thrown via `raise`.

1. [**5 points**] Create a class `Room` with members e.g. `__n`, `__s`, `__e`, and `__w`, representing the four cardinal compass points, `__desc` containing a text description of the `Room`, and `__short_desc` containing an abbreviated description. Only the description and short description should be arguments to the `__init__()` function, e.g. `__init__(self, desc, short_desc)`. `__init__()` should disallow setting the description or short description to a non-`str` or the empty string, and should throw a `TypeError` and `ValueError` in these cases, respectively.

2. [**5 points**] Within the class `Room`, create methods `set_n()`, `set_s()`, `set_e()`, and `set_w()` that take other `Room` instances.

   Name: `set_`*`dir`*
   Input(s): `room` - either `None` or a `Room` that is to the given direction of this room
   Output(s): `None` (no `return` statement)
   Throw(s): `TypeError` if `room` is not `None` or an instance of `Room`

   Each `set_`*`dir`*`()` method should be reciprocal; that is, for two `Room` instances `a` and `b`, `a.set_n(b)` should, internally, call `b.set_s(a)` to maintain proper bidirectional links. `set_`*`dir`* should, in each case, unset any existing link before setting a new one. For instance, if `a.set_n(c)` is called but `a.__n` is already set to `b`, then it should call `b.set_s(None)` before setting `a.__n` to the new `Room`, `c`.

3. [**5 points**] Within the class `Room`, create methods `n()`, `s()`, `e()`, and `w()` that return the `Room` instances in the respective directions, or `None` if no room exists in that direction.

4. [**5 points**] Within the class `Room`, override the `__str__()` method.

   Name: `__str__`
   Input(s):
   Output(s): a `str` containing the room description and short description of adjacent rooms formatted nicely for the console
   Throw(s):

   `__str__()` should display the descrption of the room in a user-readable way (should be formatted, for instance, to wrap around nicely). `__str__()` should also print the short descriptions of the adjacent rooms, e.g. "To the north you see x"

5. [**3 points**] Create a class `GateGem` that has a member e.g. `color` that indicates the color of the gem. `color` should be provided as an argument to the `__init__()` method.

6. [**5 points**] Within the class `GateGem`, override the `__str__()` method.

   Name: `__str__`
   Input(s):
   Output(s): a `str` containing a description of the gem, including its color
   Throw(s):

`__str__()` should display the descrption of the gem in a user-readable way (should be formatted, for instance, to wrap around nicely).

7. [**5 points**] Within the class `GateGem`, override the `__eq__()` method.

   Name: `__eq__`
   Input(s): `other` - another object to test equality against
   Output(s): a Boolean, `True` if the gems are equal (that is, they have the same color) and `False` if they have different colors or `other` is not an instance of `GateGem`.
   Throw(s):

8. [**5 points**] Within the class `GateGem`, override the `__hash__()` method.

   Name: `__hash__`
   Input(s):
   Output(s): an `int` or `long` value representing a hash of the `GateGem` instance such that if two `GateGem`s are equal, their hashes will be equal
   Throw(s):

9. [**5 points**] Create a class `Player` that has members e.g. `inventory`, a `Set` meant to contain `GateGem` instances, and `location`, which is the `Room` the player is currently in. A `Set` is used for inventory as we want the player to only be able to hold one `GateGem` of a given color at a time.

10. [**5 points**] Within the class `Player`, create a method `inventory_str()` that returns the contents of the player's inventory as a user-readable string.
    Name: `inventory_str()`
    Input(s):
    Output(s): a `str` representing the contents of the `Player`'s inventory, formatted to be user-readable. In particular, this should make use of `str()` to call the overloaded `__str__()` method of `GateGem`. You may assume that the inventory can only contain instances of `GateGem`.
    Throw(s):

11. [**2 points**] Now that you have something you can put in it, augment `Room` with an `inventory` like that of `Player`. Also create an analogous `inventory_str()`, and update `__str__()` to report the contents of the room, if any.

12. [**3 points**] Augment `Room` with new members `__gate_n`, `__gate_s`, `__gate_e`, and `__gate_w`, each holding either `None`, representing no gate, or an instance of `GateGem`, representing a gate that is opened using another `GateGem` of the same variety.

13. [**5 points**] Further augment `Room` with new methods `set_gate_n()`, `set_gate_s()`, `set_gate_e()`, and `set_gate_w()`, that either take `None`, clearing any existing gate, or a `GateGem`, creating a gate that can be unlocked through the use of a corresponding `GateGem` instance.
Name: `set_gate_`*`dir`*
Input(s): `gem` - either `None` or an instance of `GateGem`
Output(s): `None` (no `return` statement)
Throw(s): `TypeError` if `gem` is not `None` or an instance of `GateGem`, or `AttributeError` if there is no `Room` in the given direction.

    Each `set_gate_`*`dir`*`()` method should be reciprocal; that is, for two `Room` instances `a` and `b` such that `a.n() == b` and `GateGem` instance `g`, `a.set_gate_n(g)` should, internally, call `b.set_gate_s(g)` to make the gate accessible from either side.

14. [**2 points**] Create a class `ExitRoom`, a subclass of `Room`, which will serve as an exit point of the dungeon. You may have more than one per dungeon. Other than the difference in class, the functionality is identical to that of `Room`.

15. [**40 points**] Create a function:

    Name: `main`
    Input(s):
    Output(s):
    Throw(s):

    The `main()` function, which is called when your module is loaded, should handle interaction between the program and the user. `main()` should:

    - Initalize and connect together at least **12** rooms (excluding `ExitRoom`s),
    - Create at least **3** gates of different colors and corresponding `GateGem` instances in the rooms' inventories,
    - Create the `Player` instance and set his/her initial location,
    - Use `raw_input` and printing to control the flow of the game including inventory management (ensuring a player cannot pick up more than one of the same type of gem, removing a gem from inventory once it's used), navigation (changing the

`Player`'s location), removing gates (using the relevant `set_gate_`*`dir`* to `None`), and detecting when the user has reached the `Exit` [1] and ending the game appropriately.

The prompts in `main()` should guide the user in actions he/she can take. Supported actions should include movement in a particular direction, examining the current room, picking up a `GateGem`, if present, opening a gate, etc. Find an example game session below.

16. [**20 points**] **- Extra Credit** Make the game interesting and fun to play; really the only tools you have for this are the descriptions and the way the dungeon is laid out, but you can (and are encouraged to) augment the code as you see fit to increase the possibilities beyond the framework given. This is completely subjective, and I make no apology for it.

```
You awake in a room.

Command? (?) for help: ?
n: Move north
s: Move south
e: Move east
w: Move west
look: Examine the current room
take: Pick up an item from the room
inv: Examine your inventory

Command? (?) for help: look
The room is a dingy cellar.
To the north you see a bright room.
To the south you see mist.

Command? (?) for help: west
You struggle furiously to understand the thoughts in your own head, but fail.

Command? (?) for help: w
You run headfirst into a wall and black out.
You awake in a room.

Command? (?) for help: s
You go south.
```

[1] `isinstance(player.location,ExitRoom)`

```
Command? (?) for help: look
The room is full of fog, let in from vents towards the ceiling. Roots grow
through the cracks between the stone bricks in the wall.
To the north you see a dimly-lit room.
To the east you see a gate with a blue gem embedded in it.
There is a red gem here.

Command? (?) for help: take
You take the red gem.

Command? (?) for help: look
The room is full of fog, let in from vents towards the ceiling. Roots grow
through the cracks between the stone bricks in the wall.
To the north you see a dimly-lit room.
To the east you see a solid gate with a blue gem embedded in it.

Command? (?) for help: inv
You have a red gem.

Command? (?) for help: e
You approach the gate, but it will not budge. Its blue gem hums gently.

Command? (?) for help: n
You go north.

Command? (?) for help: n
You go north.

Command? (?) for help: look
The room is brightly lit through a skylight overhead. You can hear birds.
To the south you see a dimly-lit room.
To the west you see flickering light.
There is a red gem here.
There is a blue gem here.

Command? (?) for help: take
Which would you like to take?
1: red gem
2: blue gem
Enter selection: 1
You try to take the red gem, but as it approaches the one you already have,
they shoot apart, repelled. Perhaps you can only carry one at a time.
```

```
Command? (?) for help: take
Which would you like to take?
1: red gem
2: blue gem
Enter selection: 2
You take the blue gem.

Command? (?) for help: inv
You have a red gem.
You have a blue gem.

Command? (?) for help: s
You go south.

Command? (?) for help: s
You go south.

Command? (?) for help: look
The room is full of fog, let in from vents towards the ceiling. Roots grow
through the cracks between the stone bricks in the wall.
To the north you see a dimly-lit room.
To the east you see a solid gate with a blue gem embedded in it.

Command? (?) for help: e
You approach the gate, and the blue gem from your bag flies out and collides
with the gem embedded in the gate. The gate vanishes, leaving an open
corridor.

Command? (?) for help: inv
You have a red gem.

Command? (?) for help: look
The room is full of fog, let in from vents towards the ceiling. Roots grow
through the cracks between the stone bricks in the wall.
To the north you see a dimly-lit room.
To the east you see complete darkness.

Command? (?) for help: e
You go east.

Command? (?) for help: look
```

```
It is dark. You are likely to be eaten by a grue.
To the west you see mist.

[...]

Command? (?) for help: s
You exit the dungeon and blink hard against the bright daylight streaming
in at you. Congratulations, you are free.
```