The CSC461 Python, Iterator pattern, Strategy pattern

DUE: Thursday, October 25th, at 9AM

The purpose of this assignment is to give you practice with python and the iterator and strategy patterns.

+1 Bonus points for submitted 1 days early!

+2 more bonus points for submitted 2 days early!

Overview

You will be coding a traffic simulator. To keep the project size reasonable, there will only be one street, but with several blocks. Each block only has 1 incoming, and 1 outgoing lane. Do NOT worry about cars being in the same position nor will there be a maximum capacity. Each block will be separated by a street light. You will also only add cars at the ends of the street. A car will travel the street based on its behavior which must be set using the strategy pattern. After each update, you will output the number of cars in each lane, and whether the street lights are on or off. This must be done using the iterator pattern. You will also be able to output the car's location upon request. This also must be done using the iterator pattern.

You are given a __init__.py file. You MUST copy and paste this in "as is" into your top-most package. Its purpose is to permit redirection of a file into the console input. Since I will be posting input files and output, grading will be stricter on matching the example output as you can check your work with https://www.diffchecker.com/ .

For simplicity. you will place your blocks and lights in top to bottom order on the screen. For example:

```
Entry\Exit
↑↓
Light
↑↓
Light
↑↓
Entry\Exit
```

We will first create the class diagram in class, and you must use this diagram to code your project.

Required functionality

The menu must be in the following format as the initial state:

```
0) show road
1) update
2) add car
3) show car
4) add block
5) Quit

Choice:
```

Initially, the road will be composed of a single empty block of length 0.5 miles with a speed limit of 30 mph, and no lights. The initial output with choice 0 should be:

```
Entry\Exit
Up: 0 Down: 0
Entry\Exit
```

The up and down refer to how many cars are currently traveling up and down along that one block..

Add Car

To add a block, ask the user to input:

- Which end of the road the car is to be added (0 for top and 1 for bottom)
- Whether the car is a slow driver, normal driver, or fast driver. You must implement all three with the strategy pattern.

Add an id for each car starting at 1, and increment automatically for each car added. For simplicity, there will be instant acceleration and deceleration. For example (you MUST follow this format):

```
Choice: 2
Which end: 0) top 1) bottom: 0
Which type: 0) slow 1) norm 2) fast: 1
```

Nervous driver

A nervous driver drives 5 mph under the speed limit

Normal driver

A nervous driver drives the speed limit

Aggressive driver

An aggressive driver drives 5 mph over the speed limit

Update

On an update, update all the cars and lights by thirty seconds. The following tasks must be done in the order presented:

- A light may flip, which restarts its count down.
- At each intersection, from first to last added, see if a car can move to the next block.
    - A car can move to the next block if it is at the Entry\Exit location, and is PAST the end.
        - This means that if a block is 0.5 miles, and if car is at 0.5, do not move it.
    - A car can move to the next block if it is at the light, and the light is on.
    - Only one car may move past the intersection in one update.
    - Remove them in the order they were added to the block.
- Move the cars forward according to their behavior along the block
    - Move the cars in the same order they were added to the block.
    - If a car tries to move past a light when it is off, set its position to the length of the block.
- Print out the road
    - This must be done by adding an iterator that returns the blocks and lights in top to bottom order. In theory, you can use the same function as the “show road” option.

Show Cars

To show the cars:

1. Output the block number starting at 0 at the top
2. Then the cars location for each block
    a. In order they were added
    b. First all cars going up
    c. Then all the cars going down

This must be done with the iterator pattern, and it should work very similar to:

```
for i, v in enumerate(streetIter):
    print("Block " + str(i) + "--------------------")
    print(v.displayCars())
```

The second print has several options, but this option should to not have direct access to the blocks underlying components.

A car must display in the following format: `ID: location`. This must be done by overriding the Car's str() function. The location of a car is its distance from the start of its lane in the block. The overall format must be

```
Block 0-----------
Up----
#: loc
…
Down---
#: loc
…
Block 1-----------
…
```

Some help: Override the str() function for a lane to return the full car section of the above, and have the iterator only return the block. Then, have the block add a function to display or return a string of the contents.

Show Road

Show road should display the road:

1. Starting with Entry\Exit
2. Then a block
3. Then a light
4. Then a block (Repeat from 2 until no blocks are left)
5. Ending with Entry\Exit

To print the block, print the number of cars heading up and the number of cars heading down. To print a light, print “On” or “Off” according to its state, followed by the time till it turns. Then reprint the menu. For example, after adding two more blocks with no cars, the result would be:

```
Entry\Exit
Up: 0 Down: 0
Off 2
Up: 0 Down: 0
Off 3.5
Up: 0 Down: 0
Entry\Exit
```

Commented [RLR1]: Edit for next time to include the time

This must be done with the iterator pattern. In other words, there should be a for loop that just calls a “print” or str() on each object.

Add Block

To add a light along with another stretch of road, first ask the user to input

- the minimal length of the time for the light to turn in minutes
- the length of the block after the light in miles
- the speed limit in miles per hour

For example (you MUST follow this format):

```
Choice: 3
Length of cycle: 1
Length of block in miles: .5
Speed limit of block in mph: 30
```

Do NOT display the road right after.

This should append the light to the bottom of the road, along with the new street length. For example, adding a new light onto the initial start state and then calling choice 0 should yield:

```
Entry\Exit
Up: 0 Down: 0
Off
Up: 0 Down: 0
Entry\Exit
```

All lights start turned off. This should append the new light onto the end of the road. Then output the new road, and then the reprint the menu.

If they do not give a legal input, output `“Invalid option”` and continue.

Input errors

Input checking will be limited to the following:

- The speed limit must be a positive integer
  - The response must be “Invalid option”
- The length of the block must be a positive float >= 0.5
  - The response must be “Invalid option”
- The length of light cycle must be a positive float that is a multiple of 30 seconds
  - The response must be “Invalid option”
- If an invalid strategy is used
  - The response should be “Invalid option, using normal driver” and then use a normal behavior
- The menu should be able to handle “e,” -1, “text” with the following comment, and then reprint the menu
  - The response should be “Invalid option”

Additional restrictions

- You must display a car by overriding the str() function
- You MUST put your code into a package named your lastName_firstName (lowercase with no prefixes or suffixes. The given __init__.py file MUST be copied and pasted in “as is” into this package.
- You may not make your member variable public unless they are constants.
- Variables should be private (as much as python allows).
- The “show car” iterator option should NOT have direct access to a block’s collections. The iterator should return one block at a time.

Grading Tiers

These tiers start with the simplest tasks, and go to the most involved. The pattern will be graded separately, since they have a less clear time when to add them. HOWEVER, to get credit for them if the lower levels are not complete, you MUST put a note in your main file header explaining how to check it.

1) Have one lane
    a. display it (just the number of cars)
    b. Add a normal car
2) Update works with one car, and it is removed when it PAST the end
    a. On update, temporarily have the car display its location to confirm it is working
    b. Car str()
    c. Technically able to add the strategy pattern at this point
3) Have one block and add a car to either end
    a. On update, temporarily have the car display its location to confirm the program is working
    b. Update works with one car in both directions
    c. Cars are removed when they are PAST the end
4) Make a function to override str() to have your one block output its car according to "show car option"
5) Add more blocks (ignoring connections)
    a. Able to add a car to both ends
    b. Update still updates blocks
    c. Technically able to add the "show cars" iterator at this point
6) Add a light with no next block
    a. Light flip is correct
    b. Works with one car going down with the car stopping at the light
    c. Car removed when stopped at or past the light and the light is on
    d. Technically able to add the display iterator at this point
7) Add multiple normal cars with no next block
    a. Only one car pulled at a time at an intersection
8) Add light with block
    a. Display properly
    b. Cars work in both direction
    c. Cars placed in next block at an on light
    d. Cars still removed at when PAST the ends

Iterator

1) Add iterator to show the blocks with their cars
2) Add iterator to handle displaying the road

Strategy

1) Add the strategy pattern to a car

You must "reasonably" complete the lower tiers before gaining points for the later tiers. By "reasonably," I can reasonable assume you honestly thought it was complete.

Submission instructions (correct submission [5 points])

1. Check the coding conventions before submission.
2. If anything is NOT working from the following rubric, please put that in the bug section of your main file header.
3. If given a file that is not supposed to be changed, I will be overwriting the file with the original when I grade.
4. List any known bugs in the main file header comment.
5. All of your Python files must use a package named your lastName_firstName (lowercase with no prefixes or suffixes)
6. Delete the out folders, then zip your package folder into ***ONE*** zip folder named your lastName_firstName (lowercase with no prefixes or suffixes). Make sure this matches your package name!

If you are on Linux, make sure you see "hidden folders" and you grab the ".idea" folder. That folder is what actually lists the files included in the project!

7. Submit to D2L. The dropbox will be under the associated topic's content page.
8. *Check* that your submission uploaded properly. No receipt, no submission.

You may upload as many times as you please, but only the last submission will be graded and stored

If you have any trouble with D2L, please email me (with your submission if necessary).

Rubric

All of the following will be graded all or nothing, unless indication by a multilevel score. You must reasonably complete the tier below before you can get points for the next tier.

Did not place the __init__.py file properly: -5 points
Missing files: -2 letter grades
Does not compile: -1 letter grade
Immediate crash: -1 letter grade
Sometimes crash (within bounds of assignment up to current tier): -half-letter grade
These are on TOP of other deductions

Item	Points
Correct submission (-3 per error)	6
Coding standard (-3 per error)	8
Followed the class OOP diagram	7
Car str() overridden properly	3
Show car's iterator does not have direct access to block's contents	4
Member variables follow access level rules	4
Handles bad input (-2 for each failure)	8
Tier 1: one lane	3
Displays car(s)	2
add a normal car	1
Tier 2: update works with one lane and car	3
Update there and taking in the correct time	1
Move car appropriate distance	2
Tier 3: one block	4
Able to add a car to both ends	2
Update moves a car on either side appropriately and removed at the end	2
Tier 4: a block outputs its car locations (start of "show cars")	4
Tier 5: more blocks without connections	7
Add a new block with or without a light	2
Still able to add a car to both ends	3
Update still works on cars at least with one block	2
Tier 6: add a light with no next block with one car	8
Light cycles properly	2
Car going down stops at a "off" light	3
Car removed on "on" light	3
Tier 7: light with multiple cars (All of tier 8 still work, and cars pulled one at a time)	2
Tier 8: light with block	14
Displays road properly	3
Cars work in both directions	3
Cars placed in next block on "on" light	4
Cars removed at ends	4
Iterator	10
Display road/street done with iterator	5
Display cars down with an iterator	5
Car Strategy	5
Total	100