MELBOURNE BUSINESS SCHOOL (The University of Melbourne)
MBUSA90500 Programming, Module 2 2020
Assignment
Final submission before 9am 30 April 2020.

1 Updates

Updates will go here as the Assignment evolves.

- April 19th - Initial FAQ added.

2 Objectives

This project is designed to give your syndicate experience in software engineering, Python programming and simple optimisation problems. In particular, you should experience the following.

- Python programming - every member of the syndicate must contribute code to the final solution.
- Software design - the process of breaking a task into pieces and assigning each piece to a programmer in such a way that the pieces can be easily assembled.
- Software testing - each piece should be tested prior to integration, and then the whole tested once integrated.

3 Market Royale

You are to write a virtual player for a multiplayer game: Market Royale (based loosely on the Battle Royale genre of games). You will submit your player to a tournament server and it will play against other syndicates.
The rules of the game are as follows.

1. The game begins with a map that has various markets connected by roads. At each market you can buy and sell products. You begin at a random market with a purse full of gold and must traverse the map buying and selling products in an attempt to reach your goal before time runs out.
2. The goal is given randomly to you at the start of the game. It is a lower bound/limit on the number of each product you must obtain.
3. Prices at all markets vary for each game, but do not change in one game. When you arrive at a market, you gain knowledge on prices at other markets from any player that is already at the site and has researched other sites.
4. Before you can buy or sell at a market for the first time you must Research the market, even if you know the prices from other players.
5. You may not buy at a market if you have zero or less gold.
6. For every turn you are in debt (less than zero gold) you pay 10% interest on the overdrawn amount.
7. Over time some markets become Black. Buying and selling at Black Markets is allowed, but you must pay a fee of 100 gold for every turn you are in a Black Market (not neccesarily buying or selling). The turn before a market goes Black it goes Grey, giving you one turn to leave the market before incurring the Black penalty if you wish.
8. Your final score is computed as the sum of 10000 for each product where you have at least your goal amount plus any gold balance you have (including negative balances).
9. On each turn, your player can make the following moves.
   (a) MOVE to a neighbouring market (ie one connected by a road to your current location).

(b) RESEARCH a market.

(c) BUY some amount of one product.

(d) SELL some amount of one product.

(e) PASS to do nothing.

10. There are 50 turns in a game. There are 7 players per game.

4 Your player

Your player must be Python3 code that defines a class called `Player` that subclasses `BasePlayer` (provided). You may not change `BasePlayer` in any way.
The workhorse of `Player` is the method `take_turn` which takes 5 parameters.

1. A string name of your current market location.
2. A dictionary with values (price, amount) tuples at this market with product names as keys ({} if you have not researched this market): {`product:(price, amount)`}.
3. a dictionary of information from other players located at this market with keys as market names (strings) and values a dictionary with keys as product names and values as prices: {`market:`{`product:price`}}.
4. a list of market names (strings) that are Black.
5. a list of market names (strings) that are Grey.

The function `take_turn` should return a two-element tuple where the first element is an attribute of the `Command` class and the second is relevant data for the command (either `None` or a `(product name, amount)` tuple.
Note that your starting map, goal and gold are put into your `Player` object as attributes `map`, `goal` and `gold` respectively at the start of the game, so you can assume they exist. It is probably best not to change the `map` and `goal` attributes in your code, but you should keep track of your gold level. The Game does not change `self.gold` once it has started.
You can access markets connected to another by a road in the map using the following methods.

```
self.map.get_neighbours(node_name)}
        """@param node_name str name of node/location in map
           @return List of string names of markets connected to node_name by a road.
        """

self.map.is_road(n1, n2)
        """@param n1 str name of node/location in map
           @param n2 str name of node/location in map
           @return True if n2 is connected to n1 by a road.
        """
```

Here is a minimal player that you could submit to the tournament.

```
class Player(BasePlayer):
    """Minimal player."""
    def take_turn(self, location, prices, info, bm, gm):
        return (Command.PASS, None)
```

Your class can contain anything else you like, just as long as `take_turn` exists in the correct format. You might like a `__repr__` function. There is no way to pass arguments to the constructor of your Player object in this tournament, so if you define an `__init__` function it should not take arguments.
There will be a time limit on the `take_turn` function of 5 seconds. If your function takes longer than 5 seconds to return, you will miss that turn (`Command.PASS`).

5 Submitting to the tournament

The tournament will run as a server expecting you to send commands to add and delete players as JSON objects. You can ADD (and DEL) as many players as you like to the tournament, so a syndicate could have multiple players in the tournament at any one time.
The server is located at IP address 128.250.106.25 and port 5002.
You should use the following code (or similar) to send your JSON object to the server.

```
def send_to_server(js):
    """Open socket and send the json string js to server with EOM appended, and wait
       for \n terminated reply.
       js - json object to send to server
    """
    clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientsocket.connect(('128.250.106.25', 5002))

    clientsocket.send("""{}EOM""".format(js).encode('utf-8'))

    data = ''
    while data == '' or data[-1] != "\n":
        data += clientsocket.recv(1024).decode('utf-8')
    print(data)

    clientsocket.close()
```

The JSON object you send should contain four key:value pairs as follows.

- Key `"cmd"` which has the value of either `"ADD"`, `"DEL"` or `"PING"`.
- Key `"syn"` which has the value of your syndicate number.
- Key `"name"` which has the value of a team name for the leader board.
- Key `"data"` which has the value of string that is your python code.

Please delete your old players otherwise the system might grind to a halt.
You can view the leader board at `http://codemasters.eng.unimelb.edu.au/mbusa_2020_assignment/lb.html`.
If your player throws an exception, it is deleted from the tournament. You can view the exceptions at `http://codemasters.eng.unimelb.edu.au/mbusa_2020_assignment/e.html`.

6 Local Testing

Before submitting to the server you should test your code othwerwise you might crash the server. Also there is not much feedback from the server if your code goes wrong. To test locally, you can run a game containing just your players like this.

```
from Game import Game
p1 = Player()
p2 = Player()
g = Game([p1,p2], verbose=True)
res = g.run_game()
print(res)
```

7 Assessment

7.1 Within-syndicate responsibilities

It is expected that you will structure the Python code so that it is split into parts, and different members of the syndicate will write different parts of the code. Naturally you can collaborate, but it is in the best interests of the

weak coders that they be given a coding task to complete. The temptation for the strong coders to do the lot is high, but there will be coding questions about this assignment on the exam that each individual must answer. I suggest the strong coders concentrate on the design, and splitting the tasks up amongst the syndicate, aiming to allot the tasks based on syndicate member's abilities. If strong coders are looking for a challenge, then there is graph traversal and optimisation algorithms to extend you.
It is expected that at least one function is attributed (in comments) to each member of the syndicate.

7.2 Submission

The highest ranked player on the server from each syndicate as at 30 April 9am will be assessed. I will get the code from the server, and run it through a beautifier to format it and get a single file of Python. Please include comments, etc in your submission, and feel free to include large comments on your approach to defeating other players, and other strategic points, within the code.
The top 14 players (ie one from each syndicate) will be run in a single torunament (of at least 500 games) to determine the final position for marking.

7.3 Marking Rubric for a Syndicate

Component	Property	Mark
Submissions to server (socket + JSON)	At least one working player added	/3
(5 marks)	Several players added and deleted	/2
Basic approach to meeting goals at one market	Accounting for goals and gold	/2
(5 marks)	Making choices at a market	/3
Approach to moving	Visited more than one market	/1
	Accounted for Black markets	/2
(6 marks)	Optimisation based on other player info	/3
Final position in tournament (3 marks)	$3 - \lfloor position/4 \rfloor$	/3
Code readability	Variable names	/2
(13 marks)	Major comments (eg functions)	/5
	Minor/in-line comments when required	/2
	Modular design, no repeated code	/2
	Attribution of functions to particular authors	/2
	Total	/32

8 FAQ

1. **I'm ready to get started on the project, but there are so many files and things to understand - what's a good place to start?**

   My first suggestion is to implement the minimal Player class so that the game has all the classes necessary to run. Copy over the minimal player code from the specifications, and add in the appropriate import statements at the top.

Once this is done, you are ready to run your code and see that it works. Open up the Game.py and run the code from there. The code will now run a game with your single minimal player in it. Have a look at the output - the Game code will print out some output of the status of the game for each turn in the game: The current status of the map, the location and inventory of each player, and information about the commodities available at each market.

You can use that output to understand how your player is behaving in the Game, and then use it to tweak and improve your player.

2. **That's cool, but how should I understand the map that gets printed out?**

   The visual representation of the map shows each market as an alphabetical letter, and the paths between markets are displayed as dots ".".

   When a market turns grey, it shows up as a slash "/" on the map. When a market turns black, it shows up as a hash "#".

3. **Do we need to care about the distance between nodes, when deciding how to move?**

   Simple answer - not really - since any move between one market to another only takes 1 turn, irrespective of the distance. The only place where distance may matter depends on how you are tracking the closing of the circle (since one adjacent market might be really close to where you are, while another market adjacent market might be really far, and outside the circle).

4. **I looked at your code, and your circle isn't a circle, but actually a rectangle. Why is that?**

   The term "circle" actually comes from the term commonly used in the Battle Royal genre of games to mean the area that is not yet out of bounds, which is why it's used here.

5. **I am quite overwhelmed by this project, this project seems much bigger and more demanding than anything we've met so far. Furthermore, there are no Computer Science majors in my syndicate.**

   My first suggestion is to pause and have a look at the marking rubric (included with the specs) carefully. You'll notice that when you look at each point, you'll realize that the task is not as difficult as it seems.

   In addition, when you approach each task as a separate smaller problem, you'll be much less overwhelmed (each part then kind of becomes like a grok sized problem that you've had lots of experience dealing with by now :)) And that's one of the things about Software Engineering - the breaking down of large problems into their smaller parts, and then solving those smaller part problems to solve the overall problem. So through this project, you're getting some experience around that.

   We've provided you the game code for you to use to help you debug your Player. Note that there is no expectation that you look through or understand this code - your task is to simply implement a Player class with the `take_turn()` method, and have your Player behave based on the information it is given (either at initialization, or from each call to the `take_turn()` method).

6. **About the Map object that's initialized in our Player... I notice that it isn't updating each turn. Why is that? I want to use information about the map in my code for my Player's strategy!**

   The intention of giving you the Map object was really to give it to you as a "static" map of the area - so your Player could know what markets there are, and how they are connected (held in the `Map.map_data["node_positions"]` and `Map.map_data["node_graph"]` respectively). Since this is initialized in your player and not updated by the game code each turn, it is not a "live" map that reflects the status of each market and whether they are grey or black.

   If you want your player to keep track of the state of the game itself, then I recommend creating your own data-structure(s) to manage it.

   There is a distinction that should be made between the data the Game has, as opposed to what each Player has. The Game is managing a simulation of the game itself, and you don't have access to this information. Your Player is managing its own data about how it thinks the game is progressing and using it to make decisions.

   Think of it this way - you're writing code for a robot that's going to be dropped into a random environment (your Player). Your robot has been given a map of the area, but doesn't know how the environment will behave, nor does it have any control over the environment itself (the way the circle shrinks, the way things are randomized, the location/map that is used).

However, although you don't know how the environment will behave, you know certain things about it (ie the rules of the game). For example, you know there will be a circle that closes - so you can program your robot to estimate how that closing of the circle will be done, and thus be better adapted to react to the environment.

7. **I remember hearing that there may be updates to the code or the spec, how do I know I have the latest version of these?**

   For the code, please keep an eye out on the "Assignment" section under "Modules". When the zip file with the code is updated, it will be stated there.

   The newest spec is included together with the zip file as well. To know what updates have been made to the spec, keep an eye out on the "1. Updates" section. Having said that, I will try to keep any new clarifications together in this FAQ section here.

Good luck with the project everyone! If you have any questions, do feel to reach out, and if there's anything that's worth clarifying, I will add it on here. Happy coding!

AHT April 19, 2020