IS 210 Midterm

Programming Examination

College: CUNY School of Professional Studies
Course-Name: Software Application Programming I
Course-Code: IS 210

Overview

The Midterm Programming Exam synthesizes all of the lessons learned thus-far in the course and challenges students to produce a functional program that adheres to all the standards and best-practices thus-far presented in the course.

> ***Important***
>
> The midterm programming examination has unique submission and testing guidelines unlike that of previously encountered projects. Please take a moment to read the Submission directions prior to starting work.

Instructions

Did you know that the netcode of many major triple-a games use python to negotiate connections between players? This exercise will emulate the matchmaking functionality that dynamically builds teams of players for online multiplay.

Data Source

You are expected, as part of this exam, to create your own library of test data. Test data should resemble the following structure:

```
[(username, connection_quality),
 (username, connection_quality),
 # ...
]
```

In the above `username` is a string representing the username of the user who may be assigned to a team. `connection_quality` is an integer that has a value of either a `1` or a `0` with `1` indicating a good or *valid* connection and a value of `0` indicating a poor connection.

> ***Important***
>
> While you are expected to produce your own test data, do not include the test data or testing code in the final submission.

Specifications

1. Create a new module named, `midterm.py`.
2. Create a function named `matchmaking()` that takes four parameters, in order:
    1. `players` (list): a list of tuples of usernames and connection strength for each user.
    2. `teams` (int, optional): The number of teams to build for this game, defaults to 3
    3. `min_team` (int, optional): The minimum number of players per team, defaults to 1.
    4. `max_team` (int, optional): The maximum number of players per team, defaults to `None`.
3. The `matchmaking()` function should filter out players with bad connections and produce the lists of players, segregated into the appropriate teams, eg:

    ```
    [['player1', 'player3'], ['player2', 'player4']]
    ```

    Where players 1 and 3 are on the same team and players 2 and 4 are on the same team.
4. If there are not enough players with good connections to meet the specified number of teams and the minimum size of each team the function should return `False`, otherwise, a list of lists as above.
5. Players should be distributed in a round-robin fashion, eg, in a game with four teams, the first team would consist of players 1, 5, 9, etc while the second team would consist of players 2, 6, 10, etc and so-on.
6. All teams should have the *exact* same number of players. In cases where there are more players than can be evenly distributed across all teams, the excess players should not be assigned to a team.
7. Work should be broken up into at least two (2) functions. Students may decide which component of the work they would like to break out into the second function.

Tip

A little syntactic sugar to get you started from the reference implementation starts by initializing our game and (empty) teams using a list comprehension (coming soon!) which is like a one-line for-loop. Print the result to see what it's created:

```
game = [[] for i in xrange(teams)]
```

Tip

Filter out the bad connections early to avoid accidentally having uneven teams.

Tip

If you know the number of teams and the maximum size of each team, you should be able to determine the total number of players possible in the game -- a helpful number when trying to ensure each team has the same number of members.

> ***Tip***
>
> The modulus operator is helpful in picking which team to assign

> ***Important***
>
> Assume the list of players could be thousands of persons long. For full credit you should stop processing the list after you've collected the maximum number of players for your game. Otherwise it could take too long to keep processing the loop.

Examples

```
>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=1, min_team=1, max_team=1)
[['a']]

>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=2, min_team=1, max_team=1)
[['a'], ['c']]

>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=2, min_team=1, max_team=2)
[['a', 'd'], ['c', 'e']]

>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=2, min_team=3)
False

>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=6, min_team=1)
False

>>> players = [('a', 1), ('b', 0), ('c', 1), ('d', 1), ('e', 1), ('f', 1)]
>>> matchmaking(players, teams=1, min_team=1)
[['a', 'c', 'd', 'e', 'f']]
```

Submission

To submit this exam for credit, attach your Python file directly to your Blackboard assignment submission. You may submit this exam only once.

Consistent with the grading policy of this course, late submissions will not be accepted for any credit.

Grading

Students are encouraged to review the rubric attached the exam for details regarding the grading policy of this assignment. Code that does not run as directed in its docstring will not be eligible for credit. Similarly, code must be free of PyLint violations in order to be eligible for credit. Students are encouraged to review Week Two materials regarding how to run lint tests locally in order to ensure their submissions are free of violations.

This assignment does not ship with unit tests or an automated test site, however, code will be run under several expected scenarios to confirm that it performs the expected function.

Additionally, students are reminded that adherence to the Google Python Style Guide, DRY principle, and the Once-and-Only-Once ideology are expected components of all student submissions.

Plagiarism

Students are reminded that all forms of plagiarism including the copying of the work of others and/or academic misrepresentation are violations of University ethics codes and carry heavy penalties.

All assignments in this course are evaluated for possible plagiarism violations.