Project

Python - M2

October 7, 2024

Objective

The objective of this project is to create a Python package that simulates a fantasy sports league, complete with teams, players, and simulated games. The focus is on applying the following concepts: package creation, classes, exceptions, randomness, and dictionaries.

Make sure to keep your code clean, add comments, and organize you code in functions when you can. Respect the PEP8 and PEP257 style guides.

Organization

Create groups of 2 and use Github to work collaboratively (commit, pull requests, review, protect master branch, create your own branch etc.).

Project Breakdown

1. Setting Up the Github Repository and Python Environment

Task: Create a GitHub repository for version control and set up a Python environment for project consistency.

- **Create a Repository:**
  - Log into your GitHub account and navigate to the repositories section.
  - Click on `New` to create a new repository.
  - Name the repository, e.g., `fantasy_league`.
  - Set the visibility to `Private` if the project should remain confidential; otherwise, use `Public`.
  - Initialize the repository with a `README.md` file to document the project description and instructions.
  - Add a `.gitignore` file to ignore unnecessary files.

- **Create a Python Environment:**
  - Create a new conda environment for the project.
  - Activate the new environment.
  - Install the libraries you think you will need (e.g., `numpy`, `pandas`).
  - Export the environment configuration to a file:

    ```
    conda env export > environment.yml
    ```

- **Upload `environment.yml` to GitHub:**
  - Commit and push the file to the main branch.
  - This file will allow other collaborators to recreate the same environment using:

    ```
    conda env create -f environment.yml
    ```

- **Protect the Main Branch:**
  - On GitHub, go to the repository settings and navigate to `Branches`.
  - Add a branch protection rule for `main` to ensure that changes can only be made through Pull Requests (PRs).
  - Enable options like `Require pull request reviews before merging` to ensure code quality.
  - Each collaborator should create their own branch for their specific feature or bug fixes. You can put your initials in the branch name.
  - All changes should be merged into `main` only through PRs, after review.

2. Setting Up the Project

Task: Create a basic Python package structure.

- Create a main project directory, e.g., `fantasy_league/`.
- Inside this directory, create the following files:
  - `__init__.py` (can be empty, or used to import key classes).
  - `team.py`: For the `Team` class.
  - `player.py`: For the `Player` class.
  - `league.py`: For the `League` class and game simulation.
  - `main.py`: The script where the simulation will be executed.

3. Creating Classes

Task: Define the core classes: `Player`, `Team`, and `League`.

- **Player Class (player.py):**
  - **Attributes:** `name`, `skill_level`.
  - **Methods:** `__init__`, `__str__` (for easy printing of player details: "The player `insert_name` has a skill level of `insert_skill_level`.").
- **Team Class (team.py):**
  - **Attributes:** `team_name`, `players` (a list of `Player` objects).
  - **Methods:** `__init__`, `add_player()`, `remove_player()`, `__str__` (to print team information).
- **League Class (league.py):**
  - **Attributes:** `teams` (a list of `Team` objects), `scores` (dictionary to store game results: key is the team name, value is the number of points).
  - **Methods:**
    - `__init__`
    - `simulate_game()`: inputs: 2 teams, step 1) compare the mean of player skills to determine which team wins, step 2) update attribute `scores`: +3 for the winner, or +1 for both teams if it's a tie
    - `play_season()`: call `simulate_game()` for all combinations of teams
    - `get_results()`: print `scores`
  - **Note:** You can use NumPy to add some randomness to the result of the game.

Note: Create custom exceptions to be raised when errors about the attributes' type occur.

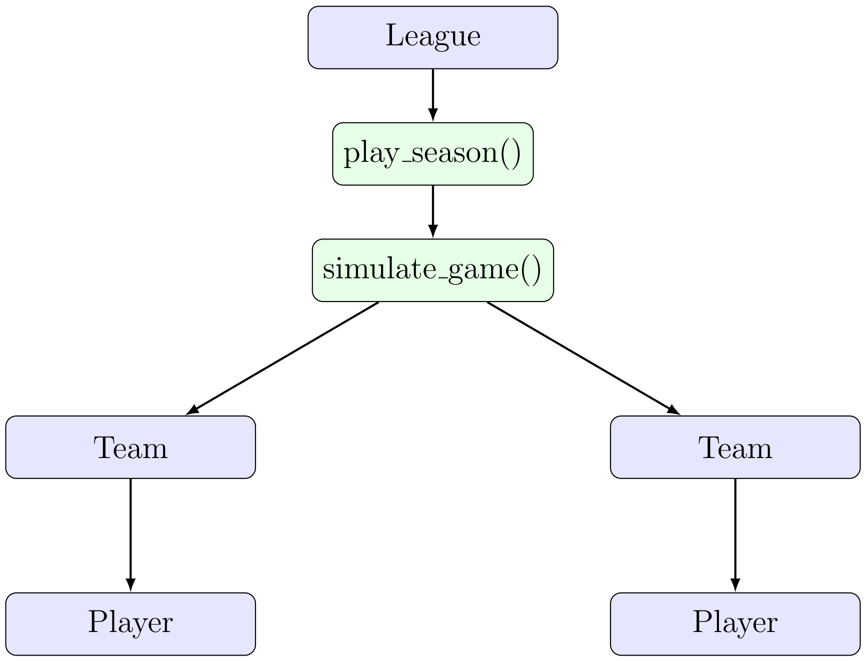
4. Simulating a Season

Task: Use the `League` class to simulate a full season of games in `main.py`.

- Create simulation settings
  - Create a file `players.csv` with variables 'Name' and 'Skill_level'. Create at least 12 players.
  - Import the csv and use it to create `Player` objects.
  - Initialize 4 `Team` objects: pick names but leave the `players` list empty.
  - Randomly assign 3 players to each team: raise an error if there are not enough players.

    - Write and call functions to organize the previous code.

- Run the simulation
    - Call `play_season()` to simulate games between all teams.
    - Track the results and determine the league champion.



5. Tests

6. Add a README