portfolio-project

For this project you will write a class called Mancala that allows two people to play a text-based version of the game (check the attached PDF file for the detailed rules).

For this board game, two players can play. As the figure shows, the player who choose the bottom red position will be player 1 and the player choose the top blue position will be player 2. Each player could only choose the pit on his side in each round: player 1 can only choose pits in red and player 2 can only choose pits in blue. The index for each pit is marked in the figure as well.

We don't require a GUI for this project and all the input/output will be in the text format. You can improve your code later on after you finished the required part to make it your own portfolio project.

Your code for the game must define the class and methods described below, but you are encouraged to define other methods or classes that may be useful for the game. All data members must be **private**.

Mancala: The Mancala object represents the game as played. The class should contain information about the players and information about the board, it must contain those methods (but may have more):

- create_player: takes one parameter of the player's name as a string and returns the player object. (You can define the player class by yourself. It will be your own design.)
- print_board: takes no parameter and will print the current board information in this format:

player1:

store: number of seeds in player 1's store

player 1 seeds number from pit 1 to 6 in a list

player2:

store: number of seeds in player 2's store

player 2 seeds number from pit 1 to 6 in a list (check the last part for examples)

- return_winner: takes no parameter.

If the game is ended, return the winner in this format:

"Winner is player 1(or 2, based on the actual winner): player's name"

If the game is a tie, return "It's a tie";

If the game is not ended yet, return "Game has not ended".

- play_game: takes two parameters, the player index (1 or 2), and the pit index (1 or 2…. or 6), both as integers. This method should follow the rules of this game including the two special rules and update the seeds number in each pit including the store. It should also check the ending state of the game and once the ending state is reached, it should follow the rules and updated the seeds number in the pit and store for both players.

If user input invalid pit index number (>6 or <=0), return "Invalid number for pit index";

If the game is ended at this point, return "Game is ended";

If the player 1 win an extra round following the special rule 1, print out “player 1 take another turn" (similar for player 2)

At the end, the method should return a list of the current seed number in this format:

[player1 pit1, player1 pit2, player1 pit3, player1 pit4, player1 pit5, player1 pit6, player1 store, Player2 pit1, player2 pit2, player2 pit3, player2 pit4, player2 pit5, player2 pit6, player2 store,]

Note: in order to test some methods in fewer steps, we may or may not follow the rules to call the two players in turns for play_game method, so your code should not enforce that. For example, we might keep calling player 1 for four times, then call player 2 twice, and then call player 1 for three times. Your python file must be named **Mancala.py**

- As a simple example, your class could be used as follows:

```
game = Mancala()
player1 = game.create_player("Lily")
player2 = game.create_player("Lucy")
print(game.play_game(1, 3))
game.play_game(1, 1)
game.play_game(2, 3)
game.play_game(2, 4)
game.play_game(1, 2)
game.play_game(2, 2)
game.play_game(1, 1)
game.print_board()
print(game.return_winner())
```

- And the output will be:

```
player 1 take another turn
[4, 4, 0, 5, 5, 5, 1, 4, 4, 4, 4, 4, 4, 0]
player 2 take another turn
player1:
store: 10
[0, 0, 2, 7, 7, 6]
player2:
store: 2
```

```
[5, 0, 1, 1, 0, 7]
Game has not ended
```

- Another test example could be:

```
game = Mancala()
player1 = game.create_player("Lily")
player2 = game.create_player("Lucy")
game.play_game(1, 1)
game.play_game(1, 2)
game.play_game(1, 3)
game.play_game(1, 4)
game.play_game(1, 5)
game.play_game(1, 6)
game.print_board()
print(game.return_winner())
```

- And the output will be:

```
player 1 take another turn
player1:
store: 12
[0, 0, 0, 0, 0, 0]
player2:
store: 36
[0, 0, 0, 0, 0, 0]
Winner is player 2: Lucy
```