Project 2 - Tactego

Due Date: Friday, November 24th, 2023 by 11:59:59 PM
Value: 90 points

This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.

```
"""
File:     FILENAME.py
Author:   YOUR NAME
Date:     THE DATE
Section:  YOUR DISCUSSION SECTION NUMBER
E-mail:   YOUR_EMAIL@umbc.edu
Description:
  DESCRIPTION OF WHAT THE PROGRAM DOES
"""
```

Project General Description

For our second project, we will be creating a simplified version of the game Stratego. If you've never heard of it, that's alright, it's a board game played between two people, playing with Red and Blue pieces. The goal of this game is essentially to capture the enemy's flag or to capture all of the enemy pieces.

However, we will not be implementing this entire game since it's probably too complicated for a cmsc201 project. So instead we'll be implementing a simplified version of the game, that I call Tactego.

Rules of the Game

1. The game of tactego is set up on a board which is a 2d grid of size length x width which will be specified at the start of the game.
2. The pieces will be specified in a file, which will have lines indicating the strength of the piece and the number of pieces with that strength.
3. Players alternate by taking turns.
4. A player selects a piece and then moves that piece.
    a. The position the player selects must be one of their pieces, you must test for this.
    b. The position that you select as the destination must be at most one place away up, down, left, right, or diagonally from the original position.
    c. The position must not contain one of the player's own pieces.
    d. If the destination contains an enemy's piece, then combat ensues.
    e. Flags cannot move.
5. Combat is determined by strength.
    a. If a higher strength piece attacks a lower **or equal** strength piece then it wins.
    b. If a lower strength piece attacks a higher strength piece, then it loses.
    c. If any piece (that can move) attacks a flag, then it captures that flag.
    d. The winner of the combat takes the position that the piece was moving into.
6. Victory for a player is when the other player has lost all of their flags.

Design Details and Flow

This is a specification for how the project should be implemented.

1. Get the size of the board.
2. Get the filename with the pieces.
3. Create the board and the pieces, and place them onto the board randomly as specified here:
    a. Use the random.shuffle method on the pieces inputted from the file in order to randomize placement of the pieces.
        i. If you have a pieces list you can simply do random.shuffle(pieces) once, for each set of pieces.
    b. For the red player, place them at the top of the board, starting at position (0, 0) and going to (0, width - 1) then go to the next line (1, 0) and scan across the row. Keep going in that fashion until all of the pieces have been placed.
    c. For the blue player, place them on the bottom of the board starting at (length - 1, 0) going up to (length - 1, width - 1) before resetting to (length - 2, 0) and scanning across that row. Keep going until all pieces have been placed.
    d. The reason we're specifying this so carefully is so that we can all use a seed and generate the same placements of pieces. It will help both you and the graders to be able to test.
4. Enter the main game loop. Stay in the game loop until one of the players wins.
5. Draw the board.
6. Get the player's move.
    a. A starting position should have two coordinates separated by a space.
    b. Check that the starting position is valid, return to (a) if not.
    c. Then get the ending position also two coordinates separated by a space.

    d. Check if the ending position is valid, return to (c) if not.
7. Move the piece.
8. Determine the result of any combat
9. Return to (4) until there is a winning player.
10. Report the player who won and end the game.

Implementation Requirements

1. You must have a main function called tactego:
```
def tactego(pieces_file, length, width):
```

2. You must import random in order to use shuffle.
3. Your main block should be:
```
if __name__ == '__main__':
    random.seed(input('What is seed? '))
    file_name = input('What is the filename for the pieces? ')
    length = int(input('What is the length? '))
    width = int(input('What is the width? '))
    tactego(file_name, length, width)
```

   This will ensure that we can all have the same placement of pieces with different random seeds, which determines the output of the shuffles.
4. Other than tactego, you should implement **at least 4 additional functions**. Keep in mind that my solution has approximately 8 functions, so you shouldn't be afraid to create far more than 5 total. You won't be rewarded for smashing too much functionality into each function so think about what the job of each function is and try to have it do that one job.
5. The only global variables that you have should be constants (and technically the inputs in main at the beginning of the program that you send into the tactego function). The game board, pieces, and all of the other things that you create for this project should be local to the tactego function. This will force you to pass them as parameters to the functions that need that data.
6. Load the pieces from the file in the order that they are given there, don't sort them or do anything before you run the shuffle on them. Run shuffle twice, once on the red pieces then once on the blue pieces in that order.

7. Output the board at least once per turn so that the player can see what they're doing.
8. Ask for the starting and ending positions in separate input statements, so that we can maintain consistency for testing.

Pieces File Format

The pieces file format will be like this, each line will take one of these forms:

1. [piece strength] [number of pieces] for example 7 3 means that there are 3 pieces with strength 7.
2. F [number of flags] for instance F 1 or F 2 [there is a space between them]
3. Extra Credit: M [number of mines]
4. Extra Credit: S [number of sappers]
5. Extra Credit: A [number of assassins]

A Sample File

Here is the sample file from small_game.pieces

```
5 1
3 1
2 2
1 3
F 1
```

Here is the sample file from basic.pieces

```
10 1
9 1
8 2
7 3
6 4
5 4
3 6
2 8
1 10
F 1
```

Pieces File Location

I will place the files in

/afs/umbc.edu/users/e/r/pub/cs201/fall23/tactego/

Any file there can be copied by doing

cp /afs/umbc.edu/users/e/r/pub/cs201/fall23/tactego/* .

If you're wondering if I've added new files you can always use:

ls /afs/umbc.edu/users/e/r/pub/cs201/fall23/tactego/

If there are new files there then you can take them. I'll probably be adding some additional ones for the extra credit parts.

PYC File

As with the last project, I'll upload a runnable PYC file. Keep in mind if you're having trouble running it, that it was created on the GL server, so if you have a different version of python you may get error messages about magic number matches and it won't run.

The file can be found at:

/afs/umbc.edu/users/e/r/pub/cs201/fall23/tactego/tactego.pyc

You can run the copy command to get it to your own directory. Currently it doesn't implement the extra credit. I'll try to get the extra credit pieces files and pyc file uploaded, it'll probably be under a different name to avoid confusion.

Extra Credit Parts

We will have two extra credit bits of the project. You do not need to do the first part to do the second part, they are independent.

Assassin Piece

The rules for assassins are here and add to the rest of the rules of the game.

1) The assassin will be able to defeat any piece on the attack.
2) The assassin however is defeated by any piece if it is attacked (it is the defending piece).

Mines and Sappers

The rules for mines and sappers will add on to the rules for the rest of the game like this [different from original stratego for the purpose of simplification].

1) A sapper is a movable piece, but strength zero, it will be defeated by any piece if it is attacked. If a sapper attacks an enemy sapper then the attacker wins by the regular rules of combat.
2) A mine is a non-moveable piece which will defeat any piece in combat but then be removed from the game.
3) If a sapper attacks a mine, the mine is defeated and the sapper remains.

Submission Details

Submit the files under the following titles:
(These are case sensitive as usual.)
submit cmsc201 PROJECT2 tactego.py

Coding Standards

Coding standards can be found here.

1. At least one inline comment per function explaining something about your code.
2. Constants above your function definitions, outside of the "if __name__ == '__main__':" block.
    a. A magic value is a string which is outside of a print or input statement, but is used to check a variable, so for instance:
        i. `print(first_creature_name, 'has died in the fight. ')` does not involve magic values.
        ii. However, `if my_string == 'EXIT':` exit is a magic value since it's being used to compare against variables within your code, so it should be:
        ```
        EXIT_STRING = 'EXIT'
        if my_string == EXIT_STRING:
        ```
    b. A number is a magic value when it is not 0, 1, and if it is not 2 being used to test parity (even/odd).
    c. A number is magic if it is a position in an array, like my_array[23], where we know that at the 23rd position, there is some special data. Instead it should be
    USERNAME_INDEX = 23
    my_array[USERNAME_INDEX]
    d. Constants in mathematical formulas can either be made into official constants or kept in a formula.
3. Previously checked coding standards involving:
    a. snake_case_variable_names
    b. CAPITAL_SNAKE_CASE_CONSTANT_NAMES
    c. Use of whitespace (2 before and after a function, 1 for readability.)

Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings, lists, dicts.
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Casting int(x), str(x), float(x), (technically bool(x))
- For loops, both *for i* and *for each* type.
- While loops
    - sentinel values, boolean flags to terminate while loops
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
    - **list slicing**
- If you have read this section, then you know the secret word is: pauldron.
- String operations, concatenation +, +=, split(), strip(), join(), upper(), lower(), isupper(), islower()
    - **string slicing**
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.

- **Dictionaries**
    - creation using dict(), or {}, copying using dict(other_dict)
    - .get(value, not_found_value) method
    - accessing, inserting elements, removing elements.
    - Deleting elements from dictionaries using del.

- Using the functions provided to you in the starter code.
- Using import with libraries and specific functions **as allowed** by the project/homework.
- You may define your own new functions with new parameters
    - Single return values (you must return a single float, bool, string, int, list or dictionary, or None/not return).
- Multiple returns are allowed, for instance:
    - x, y = function_that_returns_two_things(a,b,c)
    - You can return them by: return x, y

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- break, continue
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
    - The only exception is if you use the python None, you may use the expressions "x is not None" or "x is None".
- built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip
- If you have read this section, then you know the secret word is: alacrity.
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Sample Project Output

```
linux3[139]% python tactego.py
What is the seed? asdf
What is the filename for the pieces? small_game.pieces
What is the length? 6
What is the width?
     0    1     2    3
   0 RF   R1   R2   R2
   1 R1   R1   R3   R5
   2
   3
   4 B1   B2   B1   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 1 0
Select Position to move Piece >> 2 0
     0    1    2    3
   0 RF  R1   R2   R2
   1     R1   R3   R5
   2 R1
   3
   4 B1   B2   B1   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 4 1
Select Position to move Piece >> 3 1
     0    1    2    3
   0 RF  R1   R2   R2
   1     R1   R3   R5
   2 R1
   3     B2
   4 B1        B1   B3
   5 B1   B2   BF   B5




Select Piece to Move by Position >> 1 3
Select Position to move Piece >> 2 2
     0    1    2    3
   0 RF  R1   R2   R2
   1     R1   R3
   2 R1       R5
   3     B2
   4 B1       B1   B3
```

```
   5 B1   B2  BF   B5
Select Piece to Move by Position >> 3 1
Select Position to move Piece >> 2 0
     0    1    2    3
   0 RF   R1   R2   R2
   1      R1   R3
   2 B2        R5
   3
   4 B1        B1   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 3 2
You must select a starting position with one of your pieces,
not a flag.
Select Piece to Move by Position >> 2 2
Select Position to move Piece >> 3 2
     0    1    2    3
   0 RF   R1   R2   R2
   1     R1   R3
   2 B2
   3          R5
   4 B1       B1   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 2 0
Select Position to move Piece >> 1 0
     0    1    2    3
   0 RF   R1   R2   R2
   1 B2   R1   R3
   2
   3           R5
   4 B1        B1   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 3 2
Select Position to move Piece >> 4 2
     0    1    2    3
   0 RF   R1   R2   R2
   1 B2   R1   R3
   2
   3
   4 B1        R5   B3
   5 B1   B2   BF   B5
Select Piece to Move by Position >> 5 3
Select Position to move Piece >> 4 2
     0    1    2    3
   0 RF   R1   R2   R2
   1 B2   R1   R3
   2
   3
```

```
    4 B1         B5   B3
    5 B1   B2    BF
Select Piece to Move by Position >> 1 2
Select Position to move Piece >> 2 2
     0    1    2    3
    0 RF   R1   R2   R2
    1 B2   R1
    2           R3
    3
    4 B1         B5   B3
    5 B1   B2    BF
Select Piece to Move by Position >> 1 0
Select Position to move Piece >> 0 0
     0    1    2    3
    0 B2   R1   R2   R2
    1       R1
    2           R3
    3
    4 B1         B5   B3
    5 B1   B2    BF
B has won the game
```

Approximate Rubric

I say approximate here to mean that this rubric doesn't break down specific requirements within each group, and may be modified by a few points here and there, but this should provide you with a good understanding of what we intend the value of each piece of the project implementation to be.

Requirement	**Points** (appx)
Build the map to the correct dimensions	5
Place the pieces on the map	10
Keeping track of the player's turn	5
Taking move input and doing moves correctly	20

Determining the outcome of combat	20
Checking for victory and ending the game	10
Loading the pieces from the file correctly	10
Coding Standards, Documentation, Comments, Overall Correctness	10
Assassin Piece, loading from file and combat	+5 [EC]
Mines and Sappers, loading from the file and combat	+10 [EC]