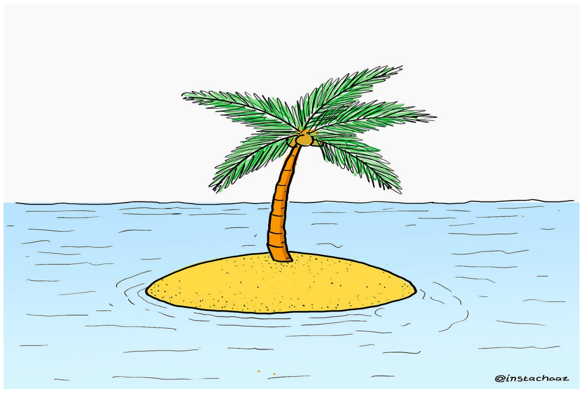
Computational Problem Solving — CSCI-603
Islands! — Lab 9



1 Introduction

You will be implementing a lab for representing and playing the pen-and-paper game, Islands. For more information about the game's rules, please refer to the problem-solving document.

2 Implementation

The main program should be named `islands.py`. The program is run on the command line as:

```
$ python islands #_of_rows #_of_columns
```

If the number of arguments is incorrect, a usage message will be displayed and the program will exit.

```
Usage: islands.py rows cols
```

If the number of arguments are correct, they are guaranteed to be valid integers greater than 0. You do not need to validate the correctness of those arguments. These arguments specify the dimension on the Islands grid.

2.1 Program Operation

You will implement a command-line interface program to play the game, all the program-user interaction will be via the standard input and output. The program must support the following user actions. The name of the command is shown on the left column. To invoke each of the commands, the user just need to type the first letter of such command, e.g. to invoke the `claim` command, the user just need to type `c` follows by the arguments that command expects.

`claim` row column	Claim the cell of the grid stored in that row and column.
`show` player	Show the list of islands owned by the specified player. The user can type this command at any time during the match.
`help`	Show all commands.
`quit`	Terminate the program. The user can type this command at any time during a match.

2.1.1 `claim` command

The `claim` command expects the position of the cell in the grid. The user must provide first the row index follows by the column index separated by a space. If the number of arguments is incorrect, the program will display the following error message and prompt again to the user to enter a valid command:

```
> c 1
RED's move...
Invalid user input!
>
```

If the number of arguments is correct, they are guaranteed to be integer values. However, the program must check whether those coordinates are valid: they must exist in the grid and that cell can be claimed (it has no owner). If the coordinates are not valid, display the following error message and prompt again to the user to enter a valid cell's coordinates:

```
> c -1 0
RED's move...
Invalid cell coordinates!
>
```

If the coordinates are valid, the cell will be claimed by the player in turn and then, the turn will be swapped to the other player. After claiming the cell, the program will display an output similar to the following:

```
> c 1 0
RED's move...

     0  1
    ------
0 |  *  -
1 |  *

Player: RED, islands: 1, Player: BLUE, islands: 1
Moves: 3, Turn: BLUE
```

2.1.2 `show` command

This command has been only added to help you debug your claim functionality and the traversal algorithm that computes the islands. This command receives as argument the player the user wants to inspect their islands. The possible values are red or blue **(case insensitive)**.

If the number of arguments is incorrect, the program will display the following error message and prompt again to the user to enter a valid command:

```
> s
Invalid user input!
> s one two
Invalid user input!
>
```

If the number of arguments is correct, the program will check if the argument is a legitimate player in the game. Otherwise, it will display the same error message from above:

```
> s white
Invalid user input!
```

If the argument is a valid player, the program will print every island currently owned by that player.

```
> c 2 1
BLUE's move...

      0  1  2
    ---------
 0 |  *     -
 1 |     -  *
 2 |  -  *

Player: RED, islands: 2, Player: BLUE, islands: 1

RED wins 2 to 1!
> s BLUE
Player: BLUE, islands: 1
Island[size:3]
    (0,2)(1,1)(2,0)
> s red
Player: RED, islands: 2
Island[size:1]
    (0,0)
Island[size:2]
    (1,2)(2,1)
>
```

Notice that the order in which the cells of an island are printed is irrelevant, printing `(1,2)(2,1)` or `(2,1)(1,2)` is equally valid. Your program doesn't have to print the cells in the same order shown here nor in the output files.

3 Design

You may design your solution as you wish. However, make sure your solution is object-oriented and breaks the problem down between various classes and methods. Minimally, you should have a class to reresent the vertices, another to represent the "brains of the game" as well as a class that reads in the user input and plays the game.

You are free to use the graph code from the lecture code. You can modify any of the code provided to suit your needs.

4 Input and Output files

You have been provided with some test files to help you verify the correctness of your implementation. Download those files from here. Here is a brief description of what is provided to you:

1. **input**: A folder containing sample files of various Islands matches.
2. **output**: A folder containing the expected output when the program run with the **input** files as `Redirect input`.

The format of the input files name is `input#_#_#.txt`. The first number is the file identifier, the second and third numbers are the number of rows and columns you should be entering as command-line arguments. For example, `input2_1_2.txt` should be tested with a grid of dimension 1x2, 1 row and 2 columns as command-line arguments.

Notice that developing your program to work with these test files is not enough; you should create your own test files to verify it thoroughly.

5 Grading

This assignment will be graded using the following rubric:

- 20%: Problem Solving
- 10% Design: The solution is object oriented and breaks the problem down between various classes and methods.
- 60% Functionality:
  - 5% Graph is built correctly.
  - 10% Cells are claimed correctly and the graph is updated accordantly.
  - 15% Computes the number of islands and the cells in each island correctly.
  - 20% Game over condition.
  - 10% Error handling: (e.g. usage message, invalid user input)
- 10% Style: Proper commenting of modules, classes and methods (e.g. using docstring's).

Good code design is a significant graded component. Your program should not be just a "monolitic" main function. The main program should be very small and pass control into classes/methods.

Also, it is very hard to award functionality points if the graph is not built correctly or the claim functionality doesn't work properly.

6 Submission

Create a ZIP file named lab9.zip that contains all your source code. Submit the ZIP file to the MyCourses assignment before the due date (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this lab).