Tent Packing

Submission to website: March 28, 10pm

Checkoff by LA/TA: March 29, 10pm

This lab assumes you have Python 2.7 installed on your machine. Please use the Chrome web browser.

Introduction

You decided to go camping with your `N` friends over spring break. Unfortunately, you have one big tent and your lazy friends didn't bother to bring their own. To accomodate your friends, you must figure out a way to arrange sleeping bags optimally in your tent. To make matters worse, after setting it up, you realize that several spots under your tent have rocks.

Your assignment is to find a way to pack the tent with sleeping bags such that: 1. **No one is sleeping on a rock.** 2. **Every usable (non-rock) portion of the tent is being utilized.** If no such arrangement exists, you must correctly conclude that no such packing exists.

Representing Tents and Friends

Similar to the last two labs, we will use a 2-dimensional grid to represent the tent. Each rock occupies one square in this grid. So, a tent configuration is described by variables: `tentSize`, which is a list with two integers – the dimensions of the tent and `missingSquares`, which is a list (possibly empty) of the grid squares with rocks under them. Each square is represented as a list with two integers – the coordinates of the square (as before, with `[0,0]` at the top left corner, and `x` and `y` axes extending to the right and down, respectively).

For example, a configuration would be represented as:

```
tentSize = [6,3],
missingSquares = [[1,2],[4,0],[4,1]]
```

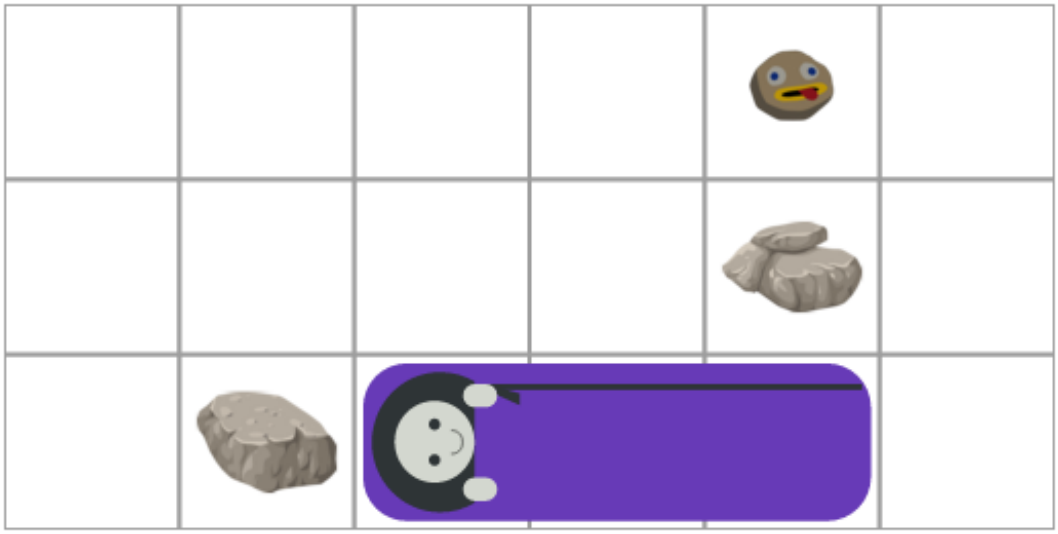
This could denote the following tent configuration -

Each person requires a `3`-by-`1` block of space to sleep. Of course, this block can be oriented either vertically or horizontally. These blocks are represented by two parameters – `anchor` and `orientation` - where `anchor` denotes the **top left square** of the `3`-by-`1` block, and `orientation` is either `0` or `1`, which corresponds to a horizontal or a vertical block, respectively.

For example, a person would be represented as a dictionary:

```
{
  "anchor": [2,2],
  "orientation": 0
}
```

In the example tent above this would correspond to:



Valid Tiling

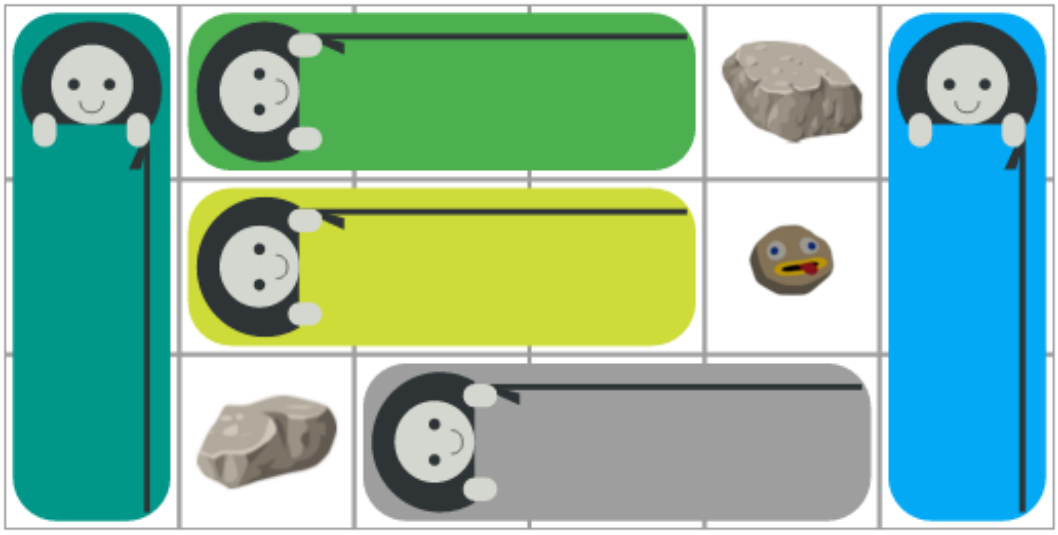
Let's say our tent has dimensions `Width` by `Height`. A valid tiling is a list of people (with

`anchor` and `orientation` values) such that: - For *each* square `[i,j]` occupied by a sleeping bag (three such squares per person): * `0 <= i < Width` and `0 <= j < Height` (i.e. each person lies entirely within the tent). * No rock exists under `[i,j]` (i.e. no person sleeps on a rock). - No two `3`-by-`1` blocks have a square in common (i.e. no two people overlap). - Every non-rock square is occupied by a person.

For example, let's say we are given the input from the **Representation** section.

```
tentSize = [6,3],
missingSquares = [[1,2],[4,0],[4,1]]
```

The following is a valid tiling for this tent with three horizontal and two vertical orientations.



The corresponding list of people (in no particular order) would look like:

```
[
  {"anchor": [1,0], "orientation": 0},
  {"anchor": [1,1], "orientation": 0},
  {"anchor": [2,2], "orientation": 0},
  {"anchor": [0,0], "orientation": 1},
  {"anchor": [5,0], "orientation": 1},
]
```

lab.py

You must implement your code in this file. You are not expected to read or write any other code provided as part of this lab. You have to correctly implement the function `find_tiling(tentSize, missingSquares)` in the file `lab.py`.

The function `find_tiling` will be invoked with the two parameters (`tentSize` and

`missingSquares`) as described in the previous section.

If there exists a complete tiling of the the non-rock squares with no overlap (more detail on this later), the function should return a list of people which results in a *valid tiling*. Each person should be represented with a dictionary (as described in the previous section) with keys `"anchor"` and `"orientation"` and valid corresponding values.

If there is no *valid tiling*, the function should return `False`.

Your code will be loaded into a tiny web server (`server.py`) which, when running, serves the Tent Packing interfaces from your very own computer acting as a web server (at http://localhost:8000 – your computer's own address at port 8000).

Run `./server.py` and go to the url http://localhost:8000 on Chrome.

In-Browser UI `./server.py`

Once your code produces output of a correct type (list of dictionaries with keys `"anchor"` and `"orientation"`), it's time to debug your logic!

You can visualize the output to help debug your code. Run `server.py` and open your browser to `localhost:8000`. You will be able to select any of the test inputs from the `./cases` folder and examine them in the browser.

You can visualize the output produced by your code by pressing the `RUN` button. This will display the tiling that your code outputs. If you output `False`, it will color the grid red.

Auto-grader (unit tester)

As before, we provide you with a `test.py` script to help you verify the correctness of your code. The script will call `find_tiling` from `lab.py` with the testcases in the `cases` folder and verify their output.

You will find that `test.py` is not very useful for debugging. We encourage you to use the UI and to write your own test cases to help you diagnose any problems and further verify correctness.

Go forth and code. Good luck. Start early!

Does your lab work? Do all tests in `test.py` pass? You're done! Submit your `lab.py` on funprog and get your lab checked off by a friendly staff member.