CS6.301: Design and Analysis of Software Systems Spring 2023

Assignment-4

Released: April 14th Deadline: April 20th

Instructions

- Submit your code along with a README.md as a zip file, named as <rollnumber>.zip.
- The readme should describe the tests you've written and what edge cases you cover for..
- The deadline is April 20th, 23:59. This is a hard deadline and no submissions will be accepted after this.
- TAs to contact : Abhijeeth, Kartik.

Introduction

For this assignment, you will need to write unit tests for the codebase provided for assignment 3. The objective of unit testing is to observe how smaller chunks of code function in isolation to ensure that they meet the expectations set for them. These expectations are not only in terms of the correctness of the value that is output, but also things such as the type of the output, the fine grained structure of the output, and the states of any external variables being modified by the code.
In essence, code that passes the unit tests written for it, shouldn't result in bugs down the line. Or in other terms, any buggy code should be caught by the unit tests written for it.

Functionality to test

For the purpose of this assignment, you will need to write tests for two different units of the provided codebase. Specifically:

- King.move(self, direction, V)

You can assume that the output given by the provided codebase is *exactly* the required output. When testing you can expect any kind of output, or any kind of error to occur and you are expected to handle them accordingly.

Testing Format:

In order to create unit tests you are required to use the 'unitttest' module in python. You need to have a test.py' file which upon running will execute your unit tests on the codebase it is currently present in.

For evaluation, the test.py file would be kept in the ‘src’ directory of the codebase to be evaluated. You should import the relevant classes and functionality from the codebase in order to create your test cases.

The expectation of test.py is that it would be run as “python3 test.py” and it would output a “True” or “False” string to a file named ‘output.txt’ indicating whether the provided codebase passes or fails the unit tests respectively. Your final output.txt (after running it on a test codebase) should look something like:

```
True
```

Note that you will need to create the ‘output.txt’ file from your code. It should only have one line which indicates the result of the codebase it was run on.

For each test that you create within test.py, also mention (in a comment) what behavior you are testing for.

Test cases

The test cases for your unit tests will be a series of codebases. Your code will be run once per code base. There will be a total of 8 basic test cases, and 2 bonus test cases for each function. Thus your total marks for the assignment will be 8, with 2 marks available as bonus.

Scope of Test Cases

Each test case will modify ***only*** the move() function. The modified move function will make changes exclusively within the King object from which it is called. It will not make changes to the Village object provided as input.
The test cases will ***only*** make use of boundary conditions (such as the edges of the map) already present in the game to define their behavior. There will be no code along the lines of:

```
if king.position == (5,7):
        # perform unexpected behavior
```

Or

```
if king.position[1] > 4:
        # perform unexpected behavior
```

Please keep in mind that the changes made to the King class itself can be unexpected, be sure to make your test cases robust.

Note: The evaluation will be automatic, so please ensure that your filenames are correct and that the code functions appropriately. Your code would be evaluated against multiple buggy and non-buggy codebases.
The correctness of your unit tests would be determined by the number of codebases correctly classified. I.e. if your code correctly identifies whether a codebase is faulty or not for 7 of the 10 codebases (including the bonus codebases), then your score for the assignment would be 7/8.

Note 2: Directly returning either True or False without testing for anything will result in a 0 for your assignment. Other similar gimmicks such as returning a random output would also receive a 0.

Bonus

As a bonus, you may also test the king.attack_target(self, target, attack) function. The details for bonus submissions are as follows:

- Number of test codebases: 8 easy + 2 tricky
- Test codebase restrictions: Only the implementation of the attack_target() function will be modified, the changes to the implementation will result in only the target class being modified.
- Testing filename: test_bonus.py
- Output filename: output_bonus.txt

Thus if you do the bonus, you should have 2 testing files:

- test.py to test king.move()
- test_bonus.py to test king.attack_target()

You should also have two output files:

- output.txt for test.py
- output_bonus.txt for test_bonus.py

The bonus marks can cover for lost marks within the assignment and also will contribute to the 'bonus' course weightage which can cover up for lost marks elsewhere.

Similar to the king.move() function we will not make the effects of attack_target() be location specific.

Changelog

- Expected filename changed from 'unittest.py' to 'test.py' (16th April)
- Updated the example score in the note regarding automatic evaluation (16th April)
- Reduced scope of assignment, provided detail about the boundaries of testcases (18th April)
- Added king.attack_target() as bonus. (19th April)