Distributed and Outsourced Software Engineering (DOSE)

Assignment 4: Testing and API Consolidation

(10 points of 100)

Deadline: Wednesday November 21th - 5 pm (Zurich time)

1. Questionnaire

Each team should have collected the time expended in Assignment 3. Please fill in the questionnaire for Assignment 3 (API Design) here:

http://tinyurl.com/dose2012-questionnaire3

Remember:

- **Only ONE person fills in the questionnaire, providing the TEAM's data.**
- **Follow the input-format suggested at each question.**

After Assignment 4, we will ask you to fill out another questionnaire on how much time you've expended for communication and preparing the assignment. Thus, please keep track of that data throughout the duration of this assignment.

2. Testing and API consolidation

In this task you should develop test cases for your project. You will use these tests during the implementation (final project phase) to check the quality of your code. During this assignment, the tests will help you to understand the APIs of the other teams' components. You should provide feedback to the API developers on how the API could be improved.

2.1 Setup

Each team will develop tests for another team's component. For example, assuming a group with for 3 teams, you could have the following setup:

Team	Develops component	Tests component
Team 1	Logic	GUI
Team 2	GUI	AI
Team 3	AI	Logic

Your group should decide which team tests which other team's component.

2.2 AutoTest configuration

For the actual testing, you should write unit tests whenever possible. EiffelStudio has an integrated testing tool called **AutoTest**. It supports different kinds of tests (manual, extracted, generated). For this task, you should only write **manual tests**. A tutorial on how to use AutoTest and how to write manual test cases can be found here:

http://docs.eiffel.com/book/eiffelstudio/using-autotest

We also provide examples of test cases for the TicTacToe game. Check out the latest version from the repository (and read the explanatory comments in the code).

The tutorial explains how you can **tag** your test cases. In order to structure the tests of all the different groups, you should tag each test case using the pattern:

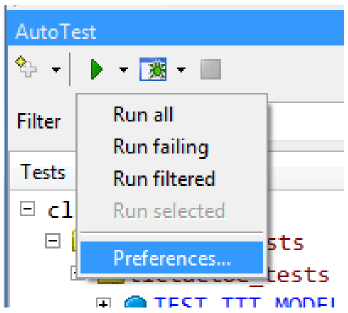
```
testing: “user/your_class_prefix”
```

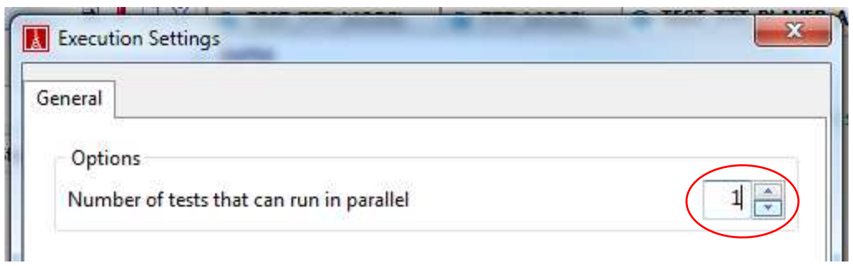
For example, all test cases of the TicTacToe game are tagged with

```
	note
		testing: "user/TTT"
	local
```

Using these tags, you can filter for the test cases relevant to your game within AutoTest. For example, all TicTacToe test cases show up when using the filter: *user/user/TTT*

Before executing the tests using AutoTest tool, you should **disable** parallel test execution (set preference value to 1):





Number of tests that can run in parallel: 1

2.2 Hand-in: Test cases and test plan

Commit all your test cases to the repository at

dose2012/src/dose_2012_tests/groupX_tests

where X is your group number. For this assignment, the tests cases can all be failing as there are no implementations yet. They tests should, however, pass by the end of the project.

Suggested Test Coverage

Assume you have requirements R_1, R_2, R_n which have been classified with different priorities (e.g. *High*, *Medium* and *Low*); and during the API design you created public routines pr_1, pr_2, pr_m and private routines r_1, r_2, r_k.

Map the routines to their corresponding requirements. Based on the requirements' priorities, you can then distinguish how well a routine should be tested:

- **S1** (*High*): public routines for requirements of high priority
- **S2** (*Medium*): public routines for requirements of medium priority
- **S3** (*Low*): public routines for requirements of low priority

Note that it is sufficient to test public routines[1].

You should write **about 2-4 test** cases for **most of the routines in S1**; at **least 1 test** case for **about 50% of the routines in S2**; and at **least 1 test** case for **about 10% of the routines in S3**.

As unit tests are not well suited for UI testing, the team in charge of UI testing should implement unit tests where possible but otherwise document the UI testing strategies (e.g. how to test manually) in a short (2-3 pages) test plan[2]. Commit your test plan to the repository at (where X is you group number):

dose2012/testplans/groupX

If the workload for the individual teams is too unequal, you can reassign team members to help out other teams.

[1] Tests of private routines are often useful for the implementer of a routine but not for the clients.

2 We do not provide a template for the test plan. Prepare a concise document that states your approach towards testing the game elements not covered by unit tests. Be instructional such that any person from the DOSE project could execute your test plan.