## MATH20622 Python Course

Lecturer: Dr Stefan Guettel | Delivery: 2019 | Nr of students: 250

---

**Marking Scheme**

The overall mark consists of lab test results (30%) and the final coursework (70%).

The deadline for submitting the coursework was 12/12/2019 at 13:00. The submission system stayed open until 13/12/2019 at 13:00.

Each coursework submission is a connect4.py text file and it has been marked semi-automatically based on four criteria:

1.) Unit testing (56%): another Python script is prepended to the original code and runs 38 unit tests, each giving a weighted mark (with 32 tests having a weight of 1, and 6 tests having a slightly higher weight depending on difficulty). The maximal number of weighted marks on unit tests is 49.
2.) Docstrings (14%): The docstrings of five selected functions are assessed and up to 2 marks given on each. Full marks could only be achieved if the docstring alone allows the function to be used by a completely uninstructed user of the module, including a list of all inputs and outputs, as well as mention of data types and possible raised exceptions.
3.) Code efficiency (20%): The code is minified using the `pyminifier` module, which removes all comments and combines print commands and dictionary definitions spanning several lines. The number of lines in the resulting code is a measure for the complexity of the code. This is put in relation to the marks gained in part 1, and scaled to a number between 0 and 1, giving a score for the code efficiency.
4.) Strategy (10%): all computer scripts enter a tournament, where each student-student combination will be played. One win amounts to 2 points and a draw yields 1 mark for each player. The overall scores are normalized to one, given an indication of the strength of the computer opponent.

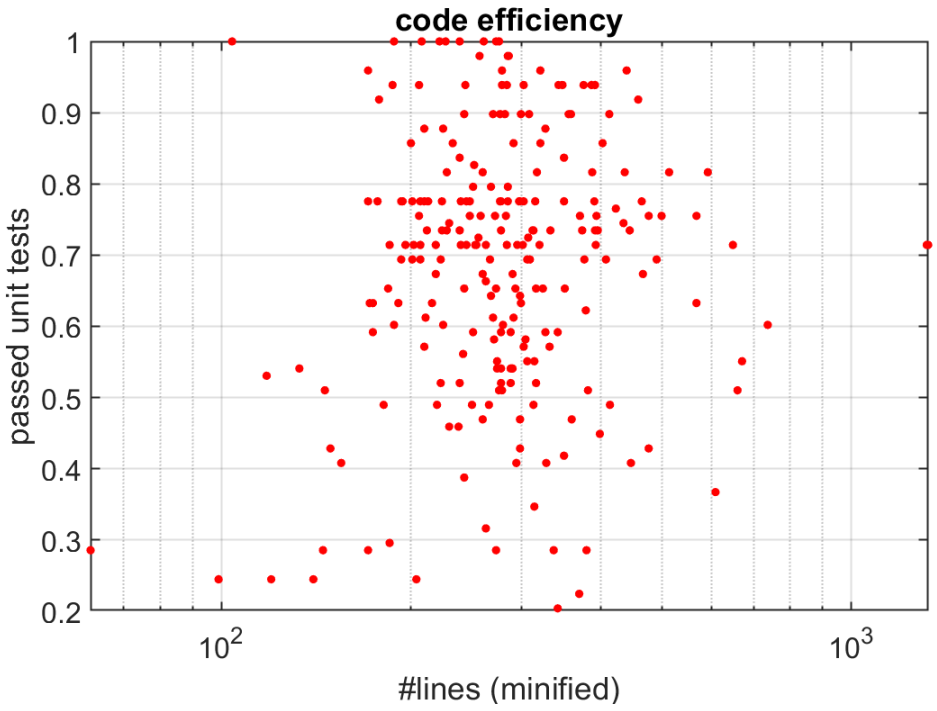
**Feedback**

Most students did a great job and some came up with very sophisticated and efficient codes! Common problems where marks have been lost were:

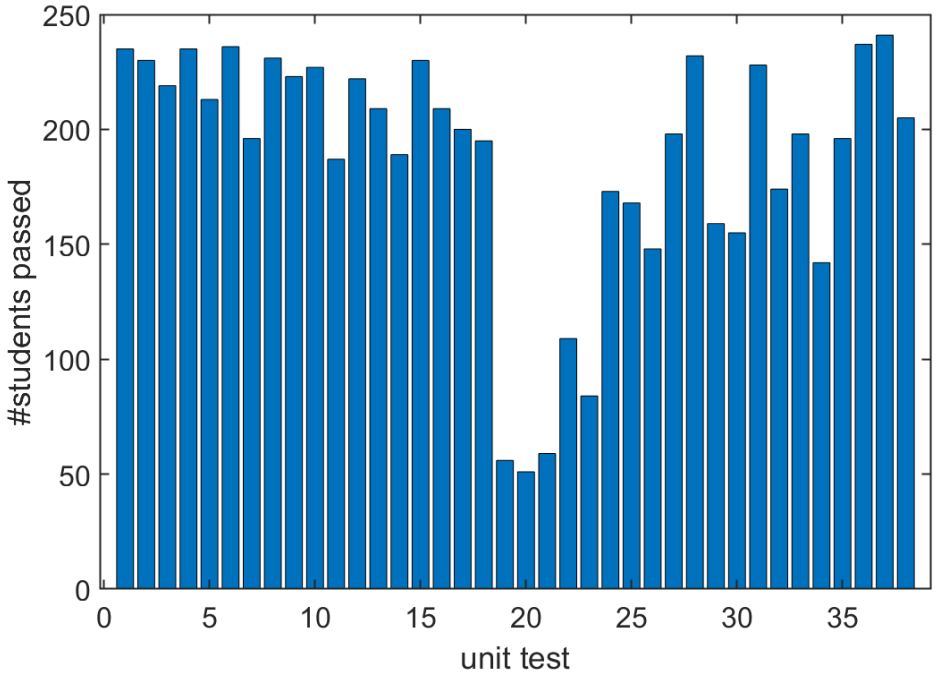
- The names of the functions or the number of input arguments do not follow exactly the coursework description. For example, `loadgame()` will be incorrect due to the missing capitalization. If a simple change in the code could correct such problems, these changes have been implemented, but often the dependencies between functions did not allow for such straightforward corrections.
- The `makeMove()` function was explicitly required to return an updated board.
- Some students represented the board array using string 'X' and 'O' values, instead of integers 1 and 2 as specified in the coursework description. In this case some unit tests naturally failed and also it was impossible for these codes to enter the strategy competition.
- Some games immediately quit after saving, which is not the intended behaviour of a ‘save’ functionality. Loading often crashed on invalid input files.
- Some programs crashed on invalid user inputs or did not use exceptions are required.
- Points have often been lost due to lacking or incomplete docstrings. It was strictly required to document all inputs and outputs of each function, including the data types of the arguments.

- The code is too long and hence inefficient. The whole coursework could be solved in about 200-250 lines of minified code (i.e., excluding empty lines, comments, docstrings, and commands broken over multiple lines). See the plot below.

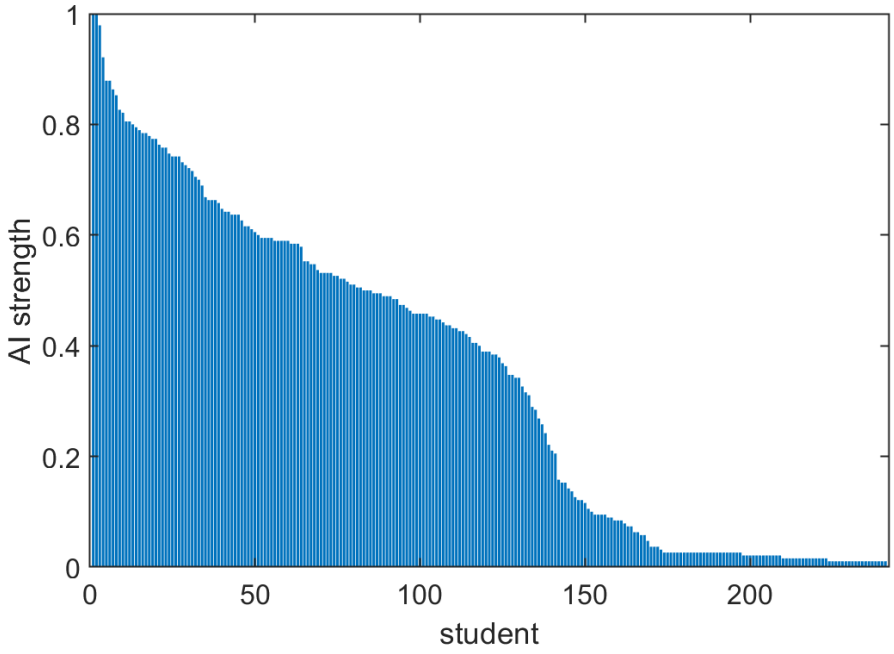
The following plot visualizes the code efficiency of all 241 submissions. Note the semi-logarithmic scale. Points in the top-left are very efficient (few lines of minified code, good functionality), whereas codes in the bottom-right are inefficient (many lines of code but little functionality).




The next graph shows how many students have passed each of the 38 unit tests. It indicates that all unit tests were solvable. The most difficult unit test, passed by only 51 students, is number 20. This test attempted to load an invalid game.txt and expected a corresponding ValueError exception.



Here are the game scores achieved by all computer opponents:




Interestingly, there is quite a large gap between the leading strategies and all others. A common feature of the best strategies seems to be a certain "look ahead" idea, where the computer would look at least for three-in-a-row, or perform additional in the future moves and then choose the best according to some evaluation function.

Finally, the overall (unscaled) mark distribution for the whole course is given below.