project 1:

simple strategies for turn-based games

solution(s) due:

May 23, 2016 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

problem specification:

task 1.1: acquaint yourself with python / numpy: download the file

```
tic-tac-toe.py
```

from the Google site that accompanies the lecture and run the script. It provides a frame and functions for the game mechanics of *tic tac toe* (*if you are not familiar with this game and its rules, look it up on Wikipedia*). Try to understand this code, play with it, and get a feeling for it; this will come in handy in task 1.3.

task 1.2: simple strategies for tic tac toe: note that both players in the provided script move entirely at random. Do something about this and implement functions that realize more intelligent moves. Proceed as follows:

1. **implement a probabilistic strategy**; have both players play many games (at least 1000) in order to create a statistic of auspicious positions on the *tic tac toe* board;

   after the whole tournament, plot a histogram of wins and draws;

   after each game in the tournament that did not end in a draw, check which player has won and determine the fields this player occupied in order to count for each field how often it contributed to a win;

   properly normalize your count data (such that they sum to one) and store them on disk; now implement a function that realizes a game move using the probabilities you just determined;

   use this function for the moves of player **X** and have player **O** move at random; start another tournament and plot the new histogram of wins and draws

2. **implement a heuristic strategy**; think of ways of evaluating the quality of a potential move;

   implement a strategy where the moving player evaluates all free positions on the board and selects the most auspicious one;

   use this function for the moves of player **X** and have player **O** move at random; start another tournament and plot the new histogram of wins and draws

task 1.3: connect four: get inspiration from looking at the code for *tic tac toe* and implement the game mechanics for *connect four* on a 6×7 board (*if you are not familiar with this game and its mechanics, look it up on Wikipedia*).
Realize proper functionality for random moves and game termination tests; have a tournament between two players moving at random (but of course according to the game rules) and try to collect statistics as to likely good moves; do you run into a problem? if so, think about what the underlying issue is; be prepared to discuss it in depth in the colloquium!
Realize a “beautiful” graphical user interface for *connect four* that allows for visualizing the progression of a game as well as for user inputs; **note:** should you have no idea whatsoever as to how to do this, simply google for “connect four python” and have a look at the many solutions you will find; you may also search for corresponding tutorials an YouTube; there is a wealth of information out there and you should use it!

task 1.4: breakout: search the Web for python realizations of the video game *breakout* (*if you are not familiar with this game and its mechanics, look it up on Wikipedia*). An example of a particularly simple implementation is *bricka*. Familiarize yourself with how *breakout* is typically implemented and play with the code you will find; for instance, see if you can increase the speed of the ball or the panel; **note:** your work on this task will not have to be presented in the colloquium, however, we will frequently consider *breakout* during the course of the semester so please familiarize yourself with it.

general hints and remarks

- Send all your solutions (code, resulting images, slides) in a ZIP archive to bauckhag@bit.uni-bonn.de

- If you are a python novice, note that there are numerous resources on the web related to python programming. Numpy and Scipy are more or less well documented and Matplotlib and pygame, too, come with tons of tutorials. Play with the code that is provided. The above tasks are easy to solve, just look around for ideas of how it can be done.
- **note:** carefully debug your code! Make sure your programs do what they are supposed to do! Programming skills are essential for this course and are expected. Flawed programs, i.e. implementations of games that violate the game rules, imply failing the course.
- Remember that you have to successfully complete all three practical projects (and the tasks therein) to be eligible to the final exam. Your grades (and credits) for this course will be decided based on the exam only, but –once again– you have to succeed in the projects in order to get there.
- Not handing in a solution implies failing the course.
- Your project work needs to be *satisfactory* to count as a success. Your code and results will be checked and your presentation needs to be convincing.
- If your solutions meet the above requirements and you can show that they work in practice, they are *satisfactory* solutions.
- A *good* to *very good* solution requires additional efforts especially w.r.t. to elegance and readability of your code. If your code is neither commented nor well structured, your solution is not good! A very good solution requires additional efforts towards the quality of your project presentation in the colloquium. Your presentation should be well timed, consistent, and convincing. Striving for very good solutions should always be your goal!