P01 – Lab Assignment

1. Group formation and organization

The lab task(s) for INTAI will be performed and graded solely within the confines of a group of **not less than 5 and not more than 7 students** (if that possibility exists). The participants of the group stay together in this constellation for the complete time of the course.

You find a spread sheet listing all participants on the course's Teams channel ("group-assignment.xlsx" under "files"). There, each inscribed student is listed with name and email-address and eventually should be assigned to a group. These assignments adhere to the rules above (i.e., 5-7 students each).

1.1 Group formation

To form the group, use verbal or electronic communication (email, Teams, etc.) and assign each member the same group ID. The ID can be any unique (reasonably short string of characters). *When the group is final, perform the corresponding changes directly in the values of the green column on the "group assignment" sheet in the Excel file on Teams*, and inform your lab supervisor.

1.2 Group organization

Once your group is fixed with respect to its members, organize yourself. This means that you will want to get to know each other with respect to your specific skills that might play a role for these labs and speak about desired roles and responsibilities. Most probably, you will designate some coordinating person to keep track of the overall process, to allocate resources, to facilitate discussions and to exchange within the group, etc.

Perform this and the next task (1.2 & 2.1) quickly, within the first morning of the block week.

2. Task: Technology demo

Until the last day of the block week, conceive, build, and present a tech demo of one aspect of AI. This consists of three parts: decision on a use case, implementation, and final presentation.

2.1 Decision on specific case (Mon morning)

Scout potential cases for which a demo could be built. The default should be to work on Task 3 below in this document. But you could also go for something of your own choice. This, then, means foremost to optimize the trade-off between relevance (how relevant would it be for the purpose of learning about and demonstrating AI?) and feasibility (what can realistically be built with the limited resources of time and labour?). The choice must be done quickly such that the remaining time can be used to work on the case – hence, it should be completed within the first morning of the block week. *When you agreed on a topic, add it to the corresponding column for your group of the "group assignment" sheet on Teams.*

Ideas for potential cases apart from task 3 below should not be limited by a pre-conceived list, but include: (a) demos that make use of open-source foundation models like large language or image generation models (see, e.g., Colossal-AI or Data Machina #190 or #193 on downscaling large models); (b) any idea from the lectures, based on available public code repositories, demonstrated on another public or private (small-scale) dataset (think: MNIST). In any case, successful technology demos are real prototypes (runnable code, real output) rather than mere theoretical concepts using "pen & paper".

It is a good idea to quickly discuss your short-list of ideas with the lab supervisor to get an outside view on how, e.g., relevancy and feasibility are seen.

2.2 Implementation (Monday-Thursday)

Building productive AI systems might require considerable resources in terms of staff and time. However, *think of a hackathon here: how can a too small team with too few resources excel anyway by making short-cuts, simplifications and assumptions that would not be tolerable in a normal project setting?* For example, prompt-based engineering methods speed up the process considerably when working with large foundation models.

All tools and accessible sources may be used for this part of the lab. Typical academic rules of credit assignment apply (e.g., reference your sources and make sure to go considerably beyond them in your main contribution).

2.3 Presentation (Friday afternoon)

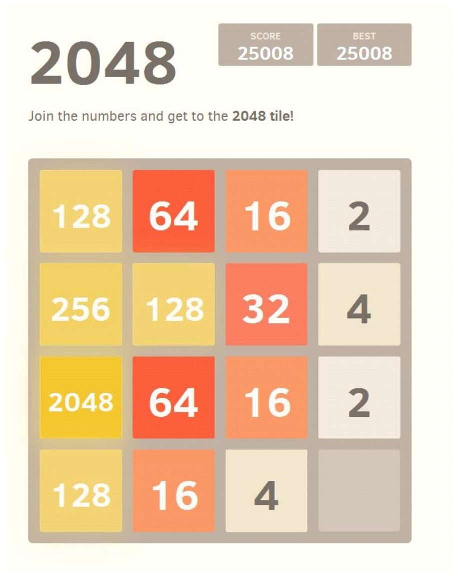
Your group must present its work in the last module time slot of the current semester (lecture and/or lab time). *You are required to present your demo in a poster session using a self-printed (and designed) A0 poster and a live demonstration of your system.* Each poster must be presented by 1-2 group members at any time, while the rest of the group walks around and discusses the other posters with their presenters.

The group is responsible to bring a computer (for the life demo) and the printed poster (A0 portrait format). An example of a poster can be found in the material package (P01_LabAssignment_material.zip) for this lab (poster-example.pptx). If you decide to build on this as a template, take care to adapt it to your specific purpose (the original purpose of the example was a specific scientific conference, where a joint publication of lecturers and students out of a project thesis in the ZHAW BSc IT programme was to be presented). A power socket, wireless internet access and table as well as a wall and means to attach the poster to the wall will be provided by the lecturers.

With respect to grading, the terms & conditions on Teams hold the authoritative information.

3. Standard technology demo case: a 2048 game-playing agent

3.1. Background



The game 2048 was developed by Italian programmer Gabriele Cirulli during a weekend, based on 1024 by Veewo Studio and conceptually similar to Threes by Asher Vollmer (you may read his story of the making & rip-offs in game development here: http://asherv.com/threes/threemails/).

The gameplay for 2048 is as follows: You can move the tiles (all of them at once) with your arrow keys. When two tiles with the same number touch, they merge into one, with the new tile having twice the old value. After each board-changing move, a new random tile spawns, having a value of 2 (90% chance) or 4 (10% chance). You win by creating a tile with the value 2048.

In this lab, you will develop a simple (part 4) as well as a sophisticated AI agent (part 5) to remote control the 2048 game running in your browser. Before, you have to learn (and previously: install) the Python programming environment that we will be using throughout this course (part 2), and configure your browser (part 3).

3.2. Installing and Using Anaconda

Continue with part 2 if you already have Anaconda installed. Other Python distributions may be used but are not supported (if you know to some degree what you are doing, that is not a problem).

Background

Anaconda is a scientific Python distribution by Continuum Analytics. Anaconda offers:

- Easy installers for Windows, MacOS & Linux with all necessary libraries included
- A good-enough and clear integrated development environment called “Spyder”

Downloading Anaconda

- Download the correct version for your system from here (several hundred MB): https://www.anaconda.com/download#downloads
    - Be sure to use the Version for Python 3.x, *not* Python 2.x
    - Take the 64-bit variant if you have a 64-bit system

Installing Anaconda

- Follow the instructions in the setup process
    - You can safely accept all standard choices
    - On Windows: If you are asked if you want to install for everyone or just yourself, choose “all users”
    - On Windows: If you are asked if Anaconda should be your standard Python 3.x environment, conform with “yes”
    - On MacOS X: if the installer says something like ‘cannot install’, choose ‘just install for me’.

Working with Spyder

Try the following things:

- Go to the interactive console tab (bottom right, tab “Console”) and use it as a calculator
- Go to `temp.py` in the code editor (left) and write your individual “hello world” script. Run it, modify it, re-run it.

Update your Anaconda Python installation

Anaconda (Python) needs to be up to date to be used for the labs accompanying this module. Please update it using either the Anaconda Launcher or the following two console commands (from Anaconda install directory):

```
conda update conda
conda update anaconda
```

See also: http://docs.continuum.io/anaconda/install/update-version/

Learning Python

As a computer scientist it should not be too difficult for you to understand short Python scripts, and from there to start writing own code.

Here are some additional resources to guide your self-study (in order of increasing sophistication):

- Beginner's cheat sheet: https://groklearning-cdn.com/resources/cheatsheet-python-1.pdf
- Programmer's cheat sheet: https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf
- Official Python 3 tutorial: https://docs.python.org/3/tutorial/
- Python for analytics: https://www.analyticsvidhya.com/learning-paths-data-science-business-analytics-business-intelligence-big-data/learning-path-data-science-python/

3.3. Configure the Chrome Browser

To play the game you should use Chrome (other solutions are possible but neither recommended nor supported).

1. Download and install Chrome from https://www.google.com/chrome/ (if it is not already installed)

2. To be able to remote control the browser, you need to start Chrome with the command line argument `--remote-debugging-port=9222`
   On Windows:
   - create a new shortcut on your desktop to the Chrome browser
   - Right click the shortcut, open "Properties"
   - In the field "Target", add `--remote-debugging-port=9222 --remote-allow-origins=*` at the end

   On Mac:

- Turn off all tabs and related tasks in Google Chrome and quit Google Chrome
- Run the following command in the terminal to set the remote debugging port and allow visit from localhost: `/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --remote-debugging-port=9222 --remote-allow-origins=http://localhost:9222`

*(Make sure to **check the text if you copy&paste** it from here: sometimes, hyphens etc. are messed up!)*

3. Install the required dependency "websocket-client" in your Anaconda Python environment:
   ```
   pip install websocket-client
   ```

4. Start Chrome with the command line argument (see above), and open the website of the game: https://play2048.co/

You are now able to run the Python code templates provided with this lab description, which will control the browser game.

3.4. Getting started with 2048

Be sure that your Firefox or Chrome browser is open (Remote Control is activated in case of Firefox, command line parameter used in case of Chrome) and correctly configured (see previous section), otherwise it will fail.

3.4.1 Running the code template `2048.py`

To run the Python script you have to open your console/terminal and write either

```
python 2048.py -b firefox
```

or

```
python 2048.py -b chrome
```

3.4.2 Modifying the code template `heuristicai.py`

To program your first own agent that will solve the game you have to modify the file `heuristicai.py`.

The goal of this task is to build an agent which is (at the minimum) better than a random player, by coming up with some rules/heuristics *apart from the algorithms treated in the lecture*. This task could also be abbreviated as *"create an AI agent manually"* – don't use AI you learned about, but hard-code your own smartness.

To do so you have to implement the method `find_best_move(board)` in a better way than it is at the moment, where it will just move the board in a random way. You are provided with a 2D list of the game state, and you have to return the direction in which the game should move.

- The parameter `board` of `find_best_move` is a numpy 2D matrix which you can access like a normal 2D array. It represents the current gamestate.
- The sample solution achieves between 7'000 and 12'000 points and in most of the cases a 512/1024 tile as its maximum.

Hints: Some ideas for useful heuristics are

- When many tiles are on the board, the chance that you can't move anymore is greatly increased; thus, one of your goals is to have the board as empty as possible.
- If tiles of big value are at the corner of the board, they don't block the merging of the "smaller" tiles.
- A move bringing two tiles with the same value next to each other is preferable over a move that won't give you this advantage.

Can you come up with some rules to master the game? C'mon!

Before you continue with the next task, make a self-assessment: How well does your agent play? How does it compare to your classmates? To results you see on the web (if your search for "2048 AI")? How does it compare to you as a human player (not only result-wise, but also if you compare the choice of moves)? What are reasons that explain what you observe? What makes implementing human game-playing (depending on your achieved results) particularly easy or hard?

3.5. AI for 2048

In this task your goal is to build a game-playing 2048 agent based on the expectimax algorithm. To do so, you will modify the script `searchai.py`. Additionally, please modify 2048.py so that it calls `find_best_move(board)` from `searchai.py`, not `heuristicai.py` as in the previous task.

To implement expectimax, you still need a good heuristic to score a current board. You can re-use ideas (or code) here from what you tried in the previous task, or implement others. Probably a big difference to your previous implementation will be that now your heuristics are combined with proven and effective systematic search capability.

[Optional: If you are not satisfied with your own heuristics, check this post for some ideas: http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048. It also provides you with other algorithms and ideas to beat the game. If you are interested, give it a try!]

Assess the performance of your solution: What makes the difference to your agent of the previous task (if any)? What differentiates it from a human player? From state of the art (see e.g. http://www.cs.put.poznan.pl/mszubert/pub/szubert2014cig.pdf)? You can compete with your peers using the following online leaderboard: https://goo.gl/meh3Ro

Hints: How to use expectimax for 2048

The main challenge in playing 2048 is the randomness of where and which kind of tile will spawn. This makes the game nondeterministic. Expectimax is a good choice for games which have a nondeterministic model of operation.

Figure 1 shows an excerpt of how expectimax will work with `DEPTH=2`. Recall that after a successful move (which alters the board), a new tile will spawn on an empty field, with the chance of a value-2 tile is 90% that of a value-4 tile is 10%. Therefore, when you have three empty fields after the move, you can have six possible outcomes as new board states after the subsequent spawning (as depicted in Figure 1).

The heuristic to score the board is only needed at the leafs of the tree. The sample solution can reach 2048 most of the time with DEPTH=2.

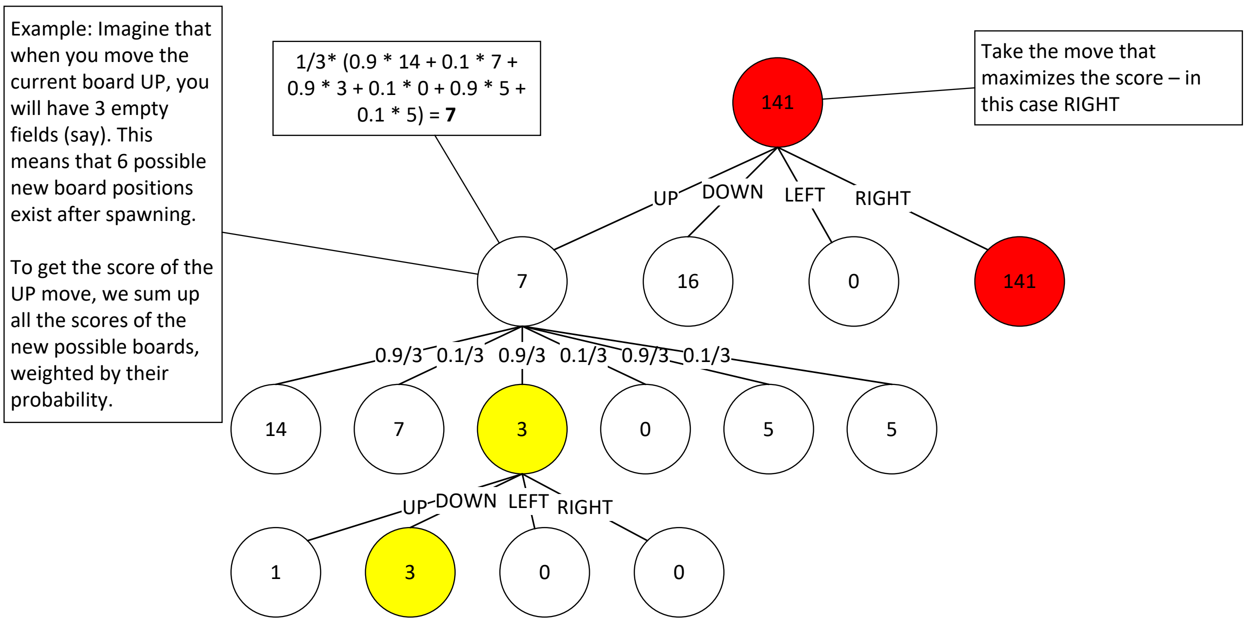

Example: Imagine that when you move the current board UP, you will have 3 empty fields (say). This means that 6 possible new board positions exist after spawning.
To get the score of the UP move, we sum up all the scores of the new possible boards, weighted by their probability.
1/3* (0.9 * 14 + 0.1 * 7 + 0.9 * 3 + 0.1 * 0 + 0.9 * 5 + 0.1 * 5) = **7**
Take the move that maximizes the score – in this case RIGHT
141
UP
DOWN
LEFT
RIGHT
7
16
0
141
0.9/3
0.1/3
0.9/3
0.1/3
0.9/3
0.1/3
14
7
3
0
5
5
UP
DOWN
LEFT
RIGHT
1
3
0
0

Figure 1: A probabilistic search tree for 2048.