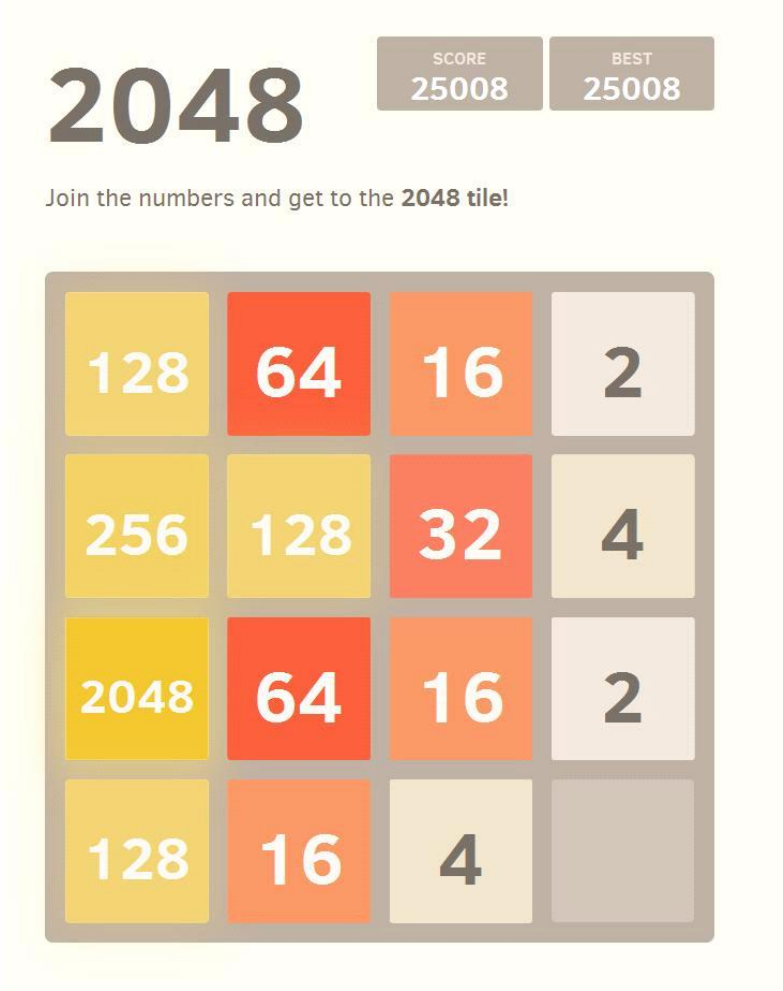
P02 – 2048 game playing agent

1. Background



The game 2048 was developed by Italian programmer Gabriele Cirulli during a week-end, 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 3) as well as a sophisticated AI agent (part 4) 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).

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.

2.1. 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”

2.2. Downloading Anaconda

- Download the correct version for your system from here (ca. 300 MB): http://continuum.io/downloads#all
  - 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

2.3. Installing Anaconda

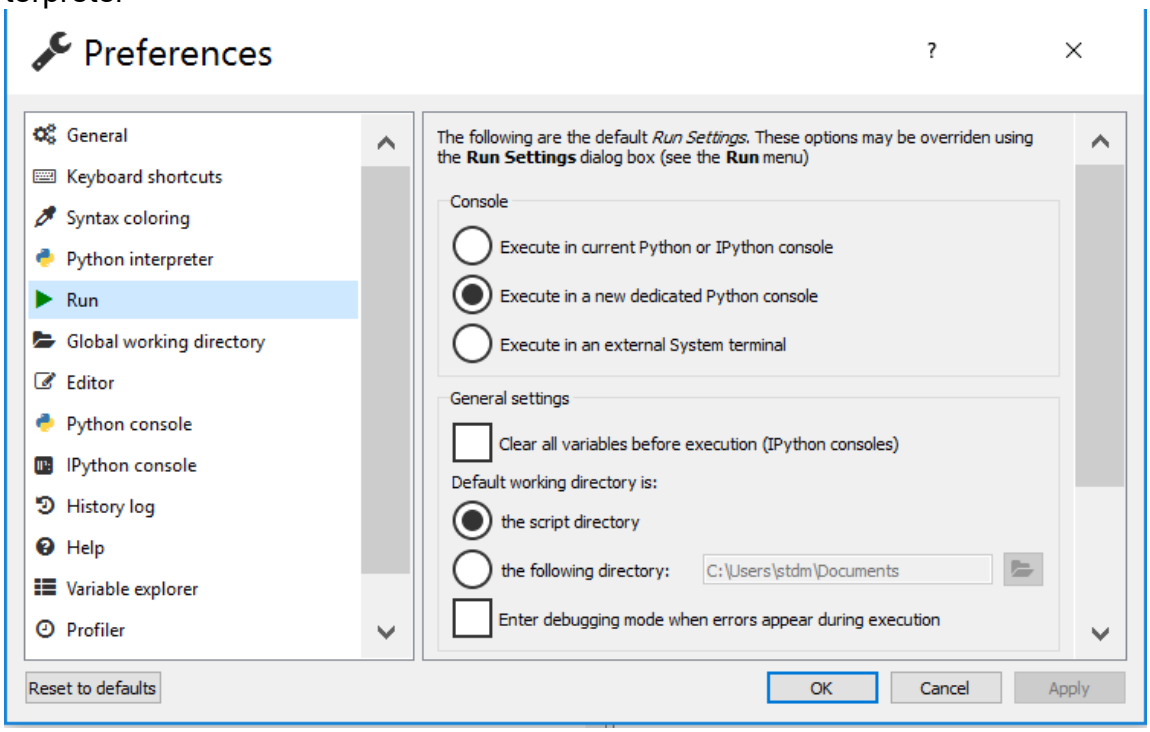
- Follow the instructions in the setup process
  - You can safely accept all standard choices
  - On Windows: If you are asked if you want to install for everyone or just yourself, choose “all users”
  - On Windows: If you are asked if Anaconda should be your standard Python 3.x environment, conform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

2.4. Configuring Spyder

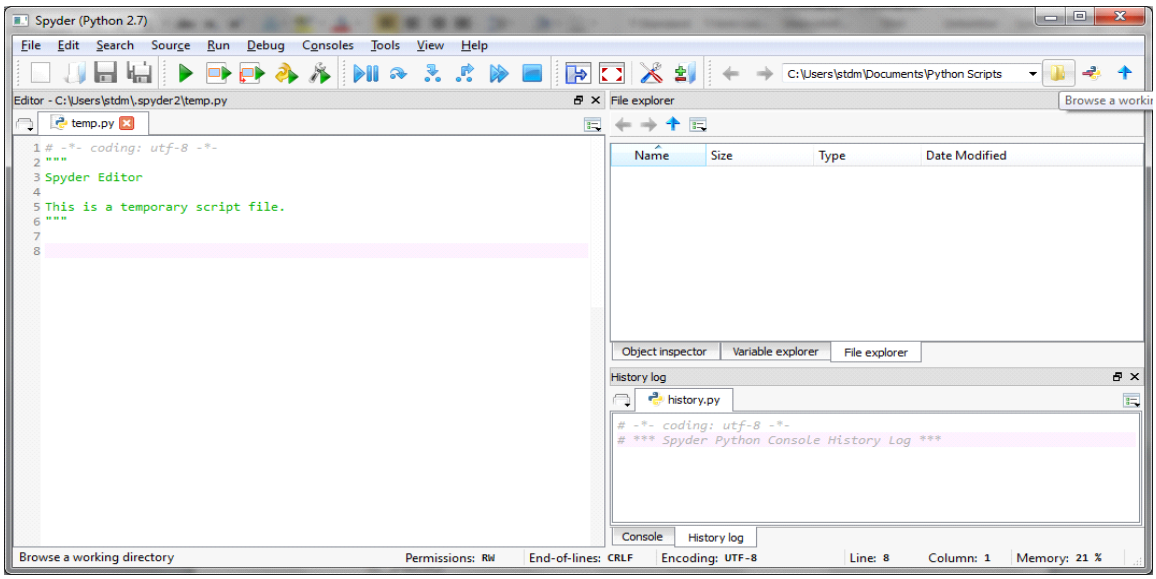
The following settings make Spyder look tidier:

- Start Spyder (on Mac: go to your home/anaconda and click on Launcher, then start Spyder)
- Go to “Tools” → “Preferences”

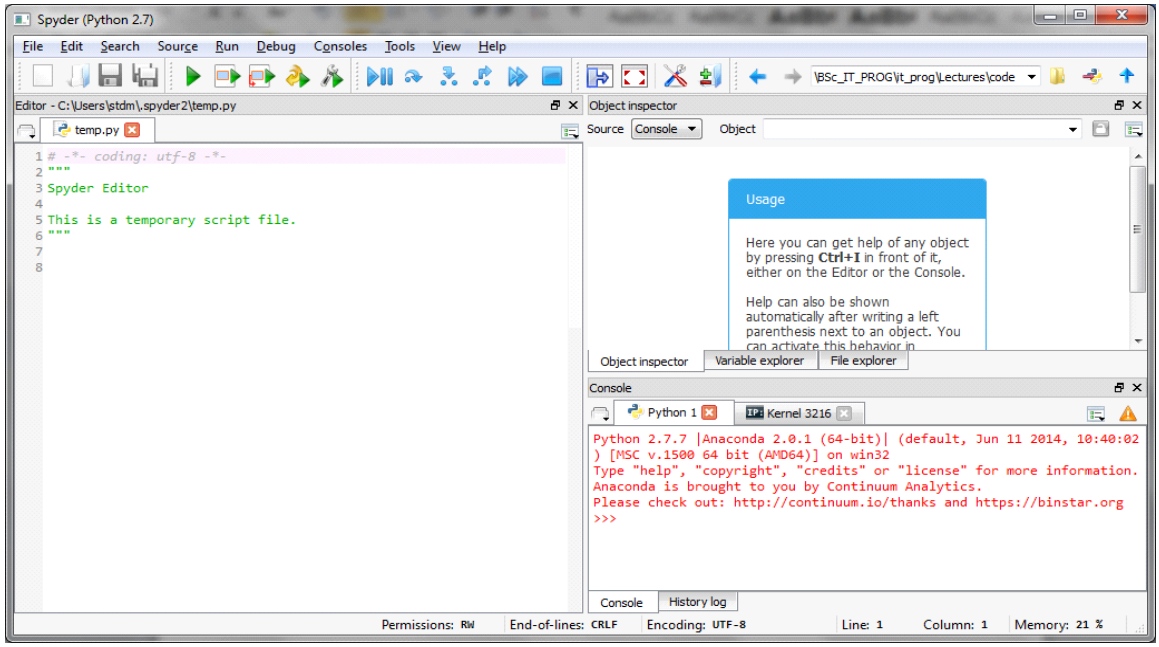
- On the “Run” page change “Console” to “Execute in a new dedicated Python interpreter”



- Close the Preferences with “ok”
- Go to “View” → “Panes” and disable the “IPython console” and “History log”
- Click the “Browse a working directory” Button at the right end of the menu bar and choose the folder where you store your scripts. This gives you easy access to all your scripts (see below)



- In the lower middle part of the window, change from the “History log” tab to the “Console”, and therein to the “Python 1” tab
  ➔ you now have immediate access to Python’s interactive console again



- In middle of the right half of the window, switch to the “File explorer” tab
  ➔ you now see a list of all the scripts in the current working directory and can easily open them

Close Spyder and start it again: All your settings should be stored and recovered.

2.5. 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.

2.6. 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#updating-from-older-anaconda-versions

2.7 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 your own code.

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

- Beginner's cheat sheet: file:///C:/Users/stdm/Downloads/cheatsheet-python-grok.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. Getting started with 2048

3.1 Installing the Remote Control plugin for Firefox and the 2048 game

Install the "Remote Control" add-on for Firefox:

- Install the Firefox developer version: https://www.mozilla.org/de/firefox/developer/
- Open Firefox Extras Add-ons Install from file and select `remote_control-1.2-fx.xpi` (ships with this lab description)
- If you experience an error because it is not signed, you have to disable signing temporarily. Go to about:config (type it into the URL window), search for `xpinstall.signatures.required` and set it to false
  (On mac you need the developer edition of Firefox for this)
- After the installation restart Firefox

To enable the plugin for the 2048 game:

- Use Firefox to open the official game website https://gabrielecirulli.github.io/2048/ (or start it locally using the `2048_original.zip` archive shipping with this lab description)

- Enable the Firefox Remote Control for this website: click the  button on the top right

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

3.2 Running the code template `2048.py`

To run the Python script you have to open your console and write `python 2048.py` (or you start it with an IDE like Spyder). Be sure that your Firefox browser is open and Remote Control is activated, otherwise it will fail.

3.2 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?

4. 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)?

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 the most of the time with DEPTH=2.

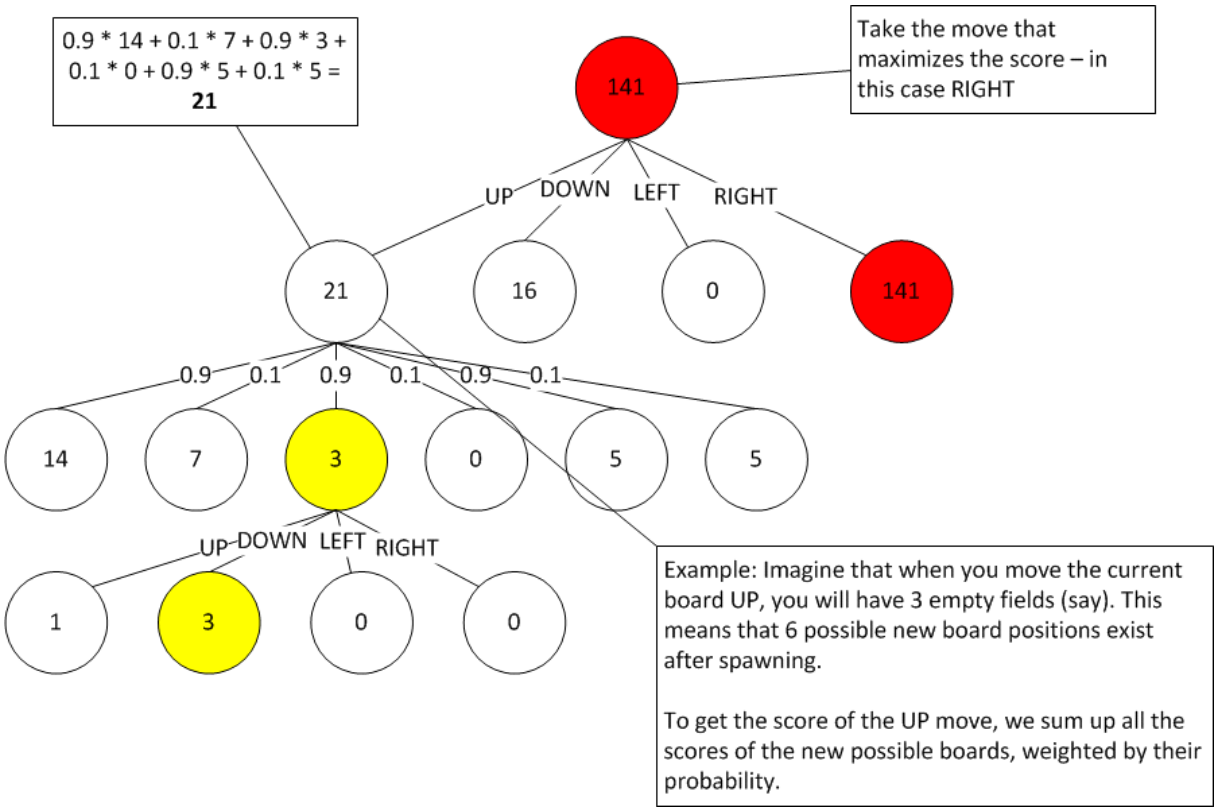


Figure 1: A probabilistic search tree for 2048.