Exercise Sheet 06 Solutions – Python Packages

Sebastian Höffner Aline Vilks

Deadline: Mon, 15 May 2017 08:00 +0200

Exercise 1: Mouse in the maze

File: `__init__.py`

```
# Errata:
# Instead of using these two statements:
#
#     import mazesolver.io
#     import mazesolver.solver
#
# It is besser to use the directory relative statements:

from . import io
from . import solver

# This basically allows to import mazesolver in the following directory
# structure:
#
#     working_directory
#     - solution
#         - mazesolver
#             - __init__.py
#             - io.py
#             - solver.py
#
# With the above mentioned imports, it's impossible to do:
#
#    import solution.mazesolver
#
# The corrected version with the dot notation (from . import...) allows this.
#
# For more information, please refer to PEP 328:
#
```

```
#     https://www.python.org/dev/peps/pep-0328/
#
```

File: io.py

```
"""This module handles the mazesolver's input and output.

Mainly this means printing to the terminal and reading the
starting configurations.
"""


import sys


def load_maze(filename):
    """Loads a maze.

    Loads a maze from a given filename.

    A maze file contains the layout of the maze as rows of numbers separated by
    spaces. The numbers encode the following:

        0: Empty space.
        1: Starting position.
        2: Wall space (not accessible).
        3: Cheese position.

    Note that only exactly one 1 and one 3 are allowed. (This is not checked.)

    Args:
        filename: The file to be read.

    Returns:
        A list containing a list per line.
        For example if the file contained:

            2 2 2 2 2 2
            2 1 0 0 3 2
            2 2 2 2 2 2

        The resulting list would look like this:

            [[2, 2, 2, 2, 2, 2],
             [2, 1, 0, 0, 3, 2],
             [2, 2, 2, 2, 2, 2]]
    """
    with open(filename, 'r') as maze_file:
        lines = maze_file.read().splitlines()
    return [[int(x) for x in line.split(' ')] for line in lines]


def print_maze(maze, file=sys.stdout):
    """Prints the maze to the file.

    Args:
        maze: The maze to print.
        file: The file to print to, defaults to sys.stdout.
    """
    for row in maze:
        print(' '.join([str(v) for v in row]), file=file)


def store_maze(maze, filename):
    """Stores a maze into a file.

    The maze is stored in the same layout as described in load_maze(filename).

    Args:
        maze: A maze as lists of lists.
        filename: The file to store the maze in.
    """
    with open(filename, 'w') as maze_file:
        print_maze(maze, maze_file)
```

File: solver.py

```
"""This module handles the maze solving."""


def solve_maze(maze, y, x):
    """Solves a maze recursively.

    The maze will be modified in-place!

    The maze should be a list of lists (each inner list representing
    a row). y and x denote the current position of the mouse, where
    y is the row index and x the column index.

    The maze solver works with backtracking:
```

```python
        If the maze is not solved:
            For all directions:
                If direction is free (i.e. the maze has a 0 in the next space):
                    Walk into the direction (set the next space to 1)
                    Solve the maze from the new position.
                    If solving was successful:
                        return True
                    Otherwise:
                        Reset the field to 0.
                Elif the cheese is found (next space is 3):
                    return True

    Args:
        maze: The maze.
        y: The mouse row.
        x: The mouse column.

    Returns:
        True if the maze was solved successfully, else False.
    """
    if not solved(maze):
        for yshift, xshift in [(-1, 0), (0, 1), (1, 0), (0, -1)]:
            if not maze[y + yshift][x + xshift]:
                maze[y + yshift][x + xshift] = 1
                success = solve_maze(maze, y + yshift, x + xshift)
                if success:
                    return True
                else:
                    maze[y + yshift][x + xshift] = 0
            elif maze[y + yshift][x + xshift] == 3:
                return True
    return False


def solved(maze):
    """Checks if the maze was solved.

    The maze is solved, if there is no 3 to be found.

    Returns:
        True if the maze has no 3.
    """
    for row in maze:
        if 3 in row:
            return False
    return True


def get_start(maze):
    """Searches for the 1 inside the maze.

    Returns:
        The row and column of the found 1.
        E.g. if 1 was in row 3 and column 4, this would return:
            3, 4
        If there is no 1 in the maze, this returns
            -1, -1
    """
    for y, row in enumerate(maze):
        for x, col in enumerate(row):
            if col == 1:
                return y, x
    return -1, -1
```

File: `solve_maze.py`

```
import os
import sys

import mazesolver


def main():
    """Searches for a possible way inside a maze.

    By default it searches the medium_maze, but if started with a program
    argument, it will use the provided maze, e.g.:

        python solve_maze.py mazes/simple_maze.txt

    Prints the loaded maze, solves the maze if possible, and prints a
    result or notification about the failure.
    """
    maze_file = os.path.join('mazes', 'medium_maze.txt')
    if len(sys.argv) > 1:
        maze_file = sys.argv[1]

    maze = mazesolver.io.load_maze(maze_file)

    print('Input')
    mazesolver.io.print_maze(maze)
```

```python
    y, x = mazesolver.solver.get_start(maze)
    if y == -1:
        print('No start given!')
        return

    success = mazesolver.solver.solve_maze(maze, y, x)

    if success:
        print('Way found!')
        mazesolver.io.print_maze(maze)
    else:
        print('No possible way.')


if __name__ == '__main__':
    main()
```

Output:

```
Input
2 2 2 2 2 2 2 2 2
2 1 0 0 0 0 0 0 2
2 0 2 0 2 2 0 2 2
2 0 2 0 2 2 0 2 2
2 0 2 0 0 2 2 2 2
2 0 2 0 2 2 2 3 2
2 0 2 2 2 0 0 0 2
2 0 0 0 2 0 0 2 2
2 0 2 2 2 0 0 2 2
2 0 0 0 0 0 0 2 2
2 2 2 2 2 2 2 2 2
Way found!
2 2 2 2 2 2 2 2 2
2 1 0 0 0 0 0 0 2
2 1 2 0 2 2 0 2 2
2 1 2 0 2 2 0 2 2
2 1 2 0 0 2 2 2 2
2 1 2 0 2 2 2 3 2
2 1 2 2 2 1 1 1 2
2 1 0 0 2 1 0 2 2
2 1 2 2 2 1 0 2 2
2 1 1 1 1 1 0 2 2
2 2 2 2 2 2 2 2 2
```