Programming Assignment 1 — Quatrominos

In the first assignment, we will consider a fictional two-player game dubbed 'Quatrominos'. It is a generalized version of the game 'Dominos' and 'Triominos'. It can also be seen as an abstraction of the 'Monkey Puzzle', from lecture 1 (see Figure 1).

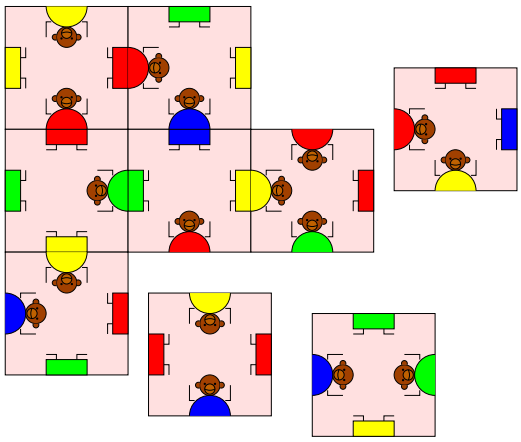

Figure 1: Illustration of the Monkey Puzzle. Quatrominos is a two-player version of an abstraction of the monkey puzzle.

Game description

The game consists of several puzzle pieces; each puzzle piece has on each side a number (instead of a color). Puzzle pieces can be placed in such a way that they are horizontally or vertically adjacent to another puzzle piece. This can only be done if the numbers of the neighboring sides are the same. Figure 2, 3 and 4 show some examples of puzzle pieces that can be connected or not.

```
+-----+  +-----+
|  2  |  |  2  |
|1   1|  |1   3|
|  2  |  |  3  |
+-----+  +-----+
```

Figure 2: Two puzzle pieces that can be connected

```
+-----+  +-----+
|  2  |  |  4  |
|1   1|  |3   3|
|  2  |  |  4  |
+-----+  +-----+
```

Figure 3: Two puzzle pieces that can not be connected

```
+-----+  +-----+
|  2  |  |  2  |
|1   1|  |3   3|
|  2  |  |  2  |
+-----+  +-----+
```

Figure 4: Two pieces that can be connected if rotated

Both players start with an equal number of puzzle pieces. Both players take turns laying out a puzzle piece according to the rules above. The players are allowed to turn their own puzzle pieces in any way they want. A player can only place a puzzle piece adjacent to an already existing puzzle piece. The first player that is able to lay out all their puzzle pieces wins the game. If a player can no longer lay out any puzzle piece (i.e., unable to make a move), that player loses.

Programming

We make the following abstraction. The players place the tiles on a 2D grid. In principle, this could be a 2D numpy array (width and height), however since each tile consists of multiple numbers, we make this a 3D numpy array (1 dimension for width, 1 dimension for height, 1 dimension for the four integers on the tile). The first player always places the first tile in the middle of this grid. As such, the first two dimensions of the numpy array always have an odd size. The final dimension of the numpy array has always static size 4. When a part of the grid does not contain a piece, it is indicated with -1. Note that if for a given `n` holds that `self.board[y][x][n] = -1`, all other values in $n = \{1, 2, 3, 4\}$ are also -1.

The goal is to write an exhaustive search algorithm that determines whether the player that is on turn can actually win the game under the assumption that both players play optimally. For a skeleton of such an algorithm, see pseudo code provided in Lecture 3.

We provided a template program. It consists of two files:

- `quatrominos.py`: The main assignment file.
- `test_quatrominos.py`: A Unit Test file.

The file `quatrominos.py` contains various functions, that are not implemented yet (in total: 8). Read the written documentation about these functions, and implement these functions. Most of these functions require less than 20 lines of code. None of the functions require more than 30 lines of code. Do not change the headers of the functions provided. You are allowed to add functions yourself, if you feel that that makes it easier. Make sure to document these consistently.

The file `test_quatrominos.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest test_quatrominos.py
```

Make sure to have the right packages installed (numpy). Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. Make sure that all other unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. In here, you can create additional unit test to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment.

Also keep in mind that all unit tests should be able to run within a matter of seconds, on any computer.

Report

Write a report in LaTeX(at most 3 pages), addressing the following points / research questions:

- Introduction: describe the game and problem. Describe a state and action.
- Draw the state-space of a small example, e.g., on a 3×3 board. You are allowed to draw this on paper and scan the result.
- Analysis of optimal method: How much time does the optimal method take on games with varying amounts of tiles? How long does it take to run on a game of 3 vs 3 tiles, and how long does it take to run a game of 4 vs 4 tiles, etc? What is the limit that you can run within 10 minutes?
- How well does the greedy approach work? How often and how much does the answer differ from the result of the optimal algorithm? Include an adversarial test-case in your report where the greedy approach does not yield an optimal move.
- Summary and Discussion. Always end a report with some summarizing remarks.

Getting Started

It seems to make sense to start with programming the print function. Note that for this, you need to access class variables, `self.player_hand` and `self.board`, indicating the hand of both players and the board, resp. You are quite free in how you want to print the board, but make sure to print it in such a way that it is clear how the tiles are oriented on the board, e.g., as done in Figure ??. Note that a `print` function always automatically includes a linebreak, which you might want to disable for certain function calls by using the argument `end`, i.e., `print('no line break', end='')`.

```
Hand player 0:
+---+
| 2 |
|2 2|
| 2 |
+---+
Hand player 1:
+---+
| 0 |
|3 1|
| 2 |
+---+
Board:
+---++---++---++---++---+
|   ||   || 1 ||   ||   |
|   ||   ||3 3||   ||   |
|   ||   || 1 ||   ||   |
+---++---++---++---++---+
+---++---++---++---++---+
|   ||   || 1 ||   ||   |
|   ||   ||2 2||   ||   |
|   ||   || 1 ||   ||   |
+---++---++---++---++---+
+---++---++---++---++---+
|   ||   || 1 || 2 || 1 |
|   ||   ||4 2||2 2||2 2|
|   ||   || 3 || 2 || 1 |
+---++---++---++---++---+
```

Figure 5: Example output of printing the game state