Assignment no. 2

Scenario: We want to write a short game of *Connect 4* (also known as *Japanese Checkers*, *Captain's Mistress*, or *Vier gewinnt*). To keep it simple, we will not use a GUI and print the game states to the terminal.

The user shall be able to specify the game size and the player token symbols in a configuration file (exercise 1). We will stick to the classic rules (see `https://en.wikipedia.org/wiki/Connect_Four`) with variable game size (exercise 2). The user should also be able to specify a number of games to be played before determining a winner (exercise 3). Finally, we will also implement a modified version of the game, where you can introduce *Pop Out*, *Power Up*, or other game variants of your choice (exercise 4).

If not explicitly stated in the exercises, the design choices for this assignment are up to you. **Only use packages mentioned in the lecture files. Write a short PDF documentation of your program (how to use it, what works/does not work).**

Exercise 5 (20 points) Read configuration from file and create game field:

The size of the playing field and the symbols for the two players shall be specified via a plain-text configuration file. The format should be

```
width=15
height=30
player1_symbol="/-\", "| |", "\-/"
player2_symbol="###", "###", "###"
```

where the order of rows is variable.

`width=w` specifies the number *w* of columns in the game. `height=h` specifies the number *h* of rows in the game. `player1_symbol=p1` and `player2_symbol=p2` specify the token symbols of the player 1 (*p1*) and player 2 (*p2*) respectively.

In order to allow for more customizable symbols, we will allow for symbols of sizes 1x1, 2x2, 3x3, and 4x4 characters. The format of the symbols will be `<"firstrow", "secondtrow", "thirdtrow">`, where the number of rows is variable.

Example 1x1 symbol:

```
player1_symbol="*"
```

should lead to a 1x1 player symbol

```
*
```

Example 3x3 symbol:

```
player1_symbol="/-\", "| |", "\-/"
```

should lead to a 3x3 player symbol

```
/-\
| |
\-/
```

Example 4x4 symbol:

```
player1_symbol="----", "....", "....", "----"
```

should lead to a 4x4 player symbol

```
----
....
....
----
```

Your program should check if the symbol for player 1 and 2 are valid (i.e. if player 1 and 2 symbol are of equal and allowed sizes).

After reading in the configurations, your program should print the game state to the console. The game state at this point is just the empty game field. Separate the columns and rows of the game field somehow. Make sure to scale the playing field such that the player symbols will fit.

The output for a game of width 7 and height 5 and 3x3 symbol size could e.g. look like this:

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
-----------------------------
```

Exercise 6 (20 points) User interaction, check game state:

Alternately ask player 1 and 2 for the column in which they want to place a new token. You may do this via the `input()` function. Check if the specified column is valid (i.e. the column exists and is not full). "Place" the token in the game field and print the new game state to the console.

After each turn, check if the game is over (if one of the two players has won or all fields are filled). Print the result if the game is over.

The output for a game of width 7 and height 5 and 3x3 symbol size after 3 turns could e.g. look like this:

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |O O|   |   |   |
|   |   |   | O |   |   |   |
|   |   |   |O O|   |   |   |
|---|---|---|---|---|---|---|
|   |   |O O| | |   |   |   |
|   |   | O |-x-|   |   |   |
|   |   |O O| | |   |   |   |
-----------------------------
```

where player 1 started with symbol `player1_symbol="O O", " O ", "O O"` and placed a token in column 2, then player 2 with symbol `player2_symbol=" | ", "-x-", " | "` placed a token in column 3, followed again by player 1 in column 3.

Exercise 7 (10 points) Play multiple rounds, write result and final state to file:

At the beginning of the game, let the user specify how many rounds should be played. Store the results and the final game states of the games and print them to a file. The player who won most of the rounds will be declared the winner.

Exercise 8 (up to 30 bonus points) Alternative game variant:

There exist many modifications of the *Connect 4* game. Implement a (reasonable) game variant of your choice (e.g. *Pop Out*, *Power Up*, or some other variant). You may also create your own game variation.

At the beginning of a game, ask the user if the original version of the game or your alternative version shall be used.

Submission: electronically via Moodle:

    https://moodle.jku.at/

Deadline: February 26th, 2018, 13:00.
Follow the Instructions for submitting homework stated on the Moodle page!