Assignment no. 3

Programming in Python I

Scenario: We want to simulate the universe.

Since this is too ambitious, we will only implement a version of *Conway's Game of Life*. We will mostly follow the specifications given in `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`.

The Game of Life, also known simply as *Life*, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.

The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties.

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The first generation is created by applying the above rules simultaneously to every cell in the inital state which is called seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

Fun fact: Conway's Game of Life is even turing complete! This means that you can build a computer within the Conway's Game of Life. You can even simulate a Conway's Game of Life within Conway's Game of Life: `https://www.youtube.com/watch?v=xP5-iIeKXE8`

The code for this assignment should be modular, following the descriptions in the exercises below. If not explicitly stated in the exercises, the design choices for this assignment are up to you. Use comments and docstrings to document your choices of how you solved a specific task and why you solved it that way.

Only use packages mentioned in the lecture files.

Exercise 10 [11 points]

In this exercise you should write a function
`read_config_file(configpath: str)`, which takes a string `configpath` as argument. `configpath` is the path of a config file that should be parsed.

Your function should extract the following entries from the config file:

- The number of iterations `n_iterations`. It is specified in a line starting with `"n_iterations:"`, followed by an arbitrary number of whitespace characters, the value of `n_iterations` as integer number, and an arbitrary number of whitespace characters. Your function should raise an `AttributeError` if the entry for `n_iterations` is missing or not convertable to an integer.
- The symbol for dead cells `dead_symbol`. It is specified in a line starting with `"dead_symbol:"`, followed by an arbitrary number of whitespace characters, one `"` character, one character that is the symbol for `dead_symbol`, another `"` character, and an arbitrary number of whitespace characters. Your function should raise an `AttributeError` if the entry for `dead_symbol` is missing or `dead_symbol` is not a single character.
- The symbol for live cells `live_symbol`. It is specified in a line starting with `"live_symbol:"`, followed by an arbitrary number of whitespace characters, one `"` character, one character that is the symbol for `live_symbol`, another `"` character, and an arbitrary number of whitespace characters. Your function should raise an `AttributeError` if the entry for `live_symbol` is missing or `live_symbol` is not a single character.
- The initial state `seed`. It is specified in a line starting with `"init_state:"`, followed by an arbitrary number of whitespace characters, a newline character, a `"` character, a string that is the value of `seed`, a newline character, a `"` character, and an arbitrary number of whitespace characters. The value of `seed` is a string that includes one or many lines of the same length seperated by newline characters. Each line in the `seed` only contains the symbols for dead or live cells, and each symbol corresponds to a cell in the `seed`. Your function should raise an `AttributeError` if the entry for `seed` is missing. Your function should raise a `ValueError` if there are other characters than the symbols for dead or live cells and newline characters in the `seed` string. Your function should raise a `ValueError` if the lines of the `seed` string do not have the same length.

Other lines in the config file should be ignored. The order of the entries in the config file can be arbitrary. Chose appropriate error messages for the different exceptions.

See file `supplements/example/example.config` for an example of a valid config file.

Your function should return a tuple with the following objects in this order:

- The value for `n_iterations` as integer.
- The symbol for dead cells `dead_symbol` as string (of length 1 because it is only one character).
- The symbol for live cells `live_symbol` as string (of length 1 because it is only one character).
- The initial state `seed` as 2D numpy array of integer, where dead cells should have value `0` and live cells should have value `1`.

You may use the file `supplements/a3_template.py` for debugging but make sure to only hand in the function `read_config_file(configpath: str)`.

Hint: Regex makes parsing the config file easier.
Hint: First convert the `seed` string to a list where each element is a line in the `seed` string. Then convert each of those strings to a list, so that you get a nested list of characters. You can then convert this nested list of characters directly to a numpy array. Numpy arrays support element-wise boolean operations, for example `my_array == 5`.

Exercise 11 [11 points]

In this exercise you should write a function `__get_next_state__(state)`, where `state` is a game state as 2D numpy array of type `numpy.int`. Live cells in `state` have value `1` and dead cells have value `0`. The function should compute the next game state from the old game state `state`.

Your function should apply the following rules to compute the next game state:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
5. **Any cells outside the boundaries of the numpy array `state` are considered dead.**

Note that the next status of each cell is updated/computed concurrently from the old state. This means that you have to create a separate numpy array to hold the new state (instead of using in-place operations on the old state array).

Your function should return the new state as 2D numpy array of type `numpy.int`. Live cells should have value `1` and dead cells should have value `0`.

You may use the file `supplements/a3_template.py` for debugging but make sure to only hand in the function `__get_next_state__(state)`.

Hint: There are many different ways to solve this task. Some examples would be:

- You can use numpy operations on the state array (e.g. you can create an integer array where you compute/store the number of neighbors using slicing).
- You can use 2 nested Python for-loops to loop over all elements in the state array and compute the next state for each cell.
- You can use 2 nested Python for-loops to loop over all elements in the state array, compute the next state for each cell, and optimize the function using numba.

Exercise 12 [8 points]

tba

Exercise 13 [5 points]

tba

Exercise 14 [5 bonus points]

tba

Submission: electronically via Moodle:

`https://moodle.jku.at/`

Deadline: For deadlines see individual Moodle exercises.
Follow the **instructions for submitting homework** stated on the Moodle page!