Home Assignment 2
Due date:25.04.19 23:30

In this task, you need to build a class that can operate a session of "game of life". Before starting you should be familiar with Game of Life, we strongly recommend reading the Wikipedia page https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.
For this work, you need to build a class that will inherit the properties of an interface named game_of_life_interface.py. You need to implement the different methods of the interface according to the restrictions.

Constructor Initial Variables:

Your class should get three values during initialization: the size of the board, starting position and rules.

- *size_of_board* is an integer bigger than 9 and smaller than 1000, i.e., an int in $\{10,\ldots,999\}$.
- *starting_position* is an integer. Only the integers defined in the paragraph Starting Position should work, if a different integer is provided the game should start with starting_position = 1.
- *rules* is a string that holds the rules of the game. for more information read: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Variations

Public Methods to Implement

All the methods are also described in the interface file.

- *return_board()*: This method returns a list of the board. The board is a two-dimensional list that every cell denotes if the cell is dead or alive. Dead will be denoted with 0 while alive will be denoted with 255.
- *update()*: This method updates the board game by the rules of the game.
- *save_board_to_file(file_name)*: This method saves the current state of the game to a file. file_name is a string that denotes the file name. The file should be a .png file, for example, file_name='1000.png'. You should use Matplotlib for this.
- *display_board()*: This method displays the current state of the game to the screen. You can use Matplotlib for this.

Starting Position:

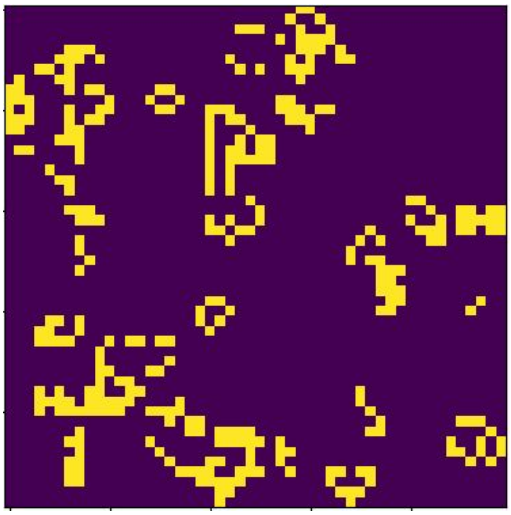
see examples in the next paragraph for more details.

- For starting position = 1, the board should be arranged randomly as follows. Each cell is `alive' with probability ½ (and `dead' with the complement probability, that is, ½ as well). you can use the np.random module.
- For starting position = 2, the board should be arranged randomly as follows. Each cell is `alive' with probability 0.8 (and `dead' with the complement probability, that is, 0.2). you can use the np.random module.
- For starting position = 3, the board should be arranged randomly as follows. Each cell is `alive' with probability 0.2 (and `dead' with the complement probability, that is, 0.8). you can use the np.random module.
- For starting position = 4, the board should start empty with a Gosper Glider Gun in top left cell at (10,10).

- For starting position = 5, the board should start empty with a Pulsar in the middle of the board.
- For starting position = 6, the board should start empty with a Grin in top left cell at (5,5).

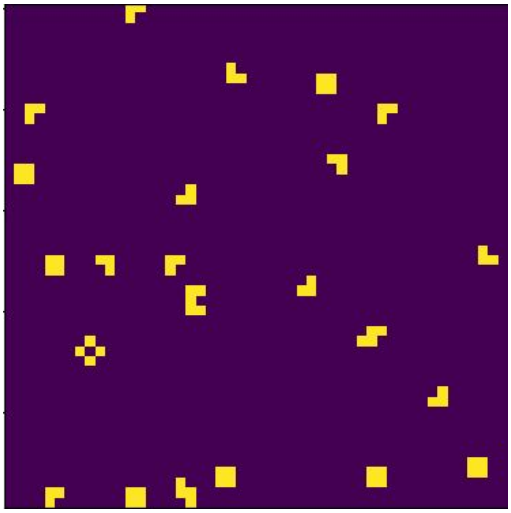
Some Examples:

running the code for 100 iterations with a *size_of_board=50, starting_position=1 and rules=B36/S23 will give us:*



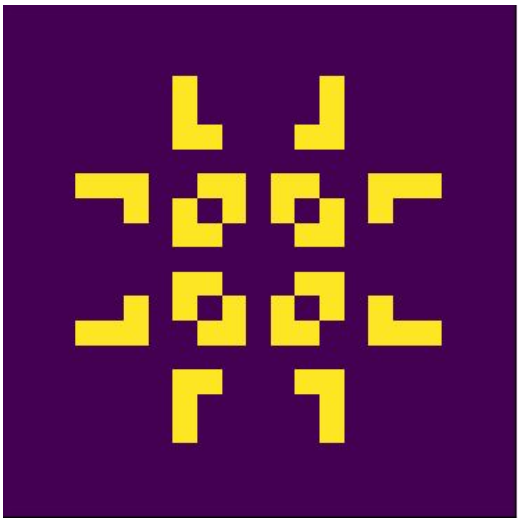
(run time was around 0.84 sec) (this example is based on a random seed, therefore don't expect it to be equal to your result)

running the code for 20 iterations with a *size_of_board=50, starting_position=2 and rules=B45/S23 will give us:*



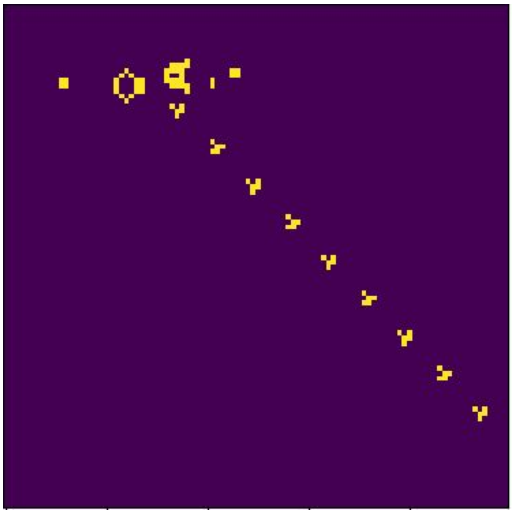
(run time was around 0.17 sec) (this example is based on a random seed, therefore don't expect it to be equal to your result)

running the code for 115 iterations with a *size_of_board=21, starting_position=5 and rules=B3/S23 will give us:*



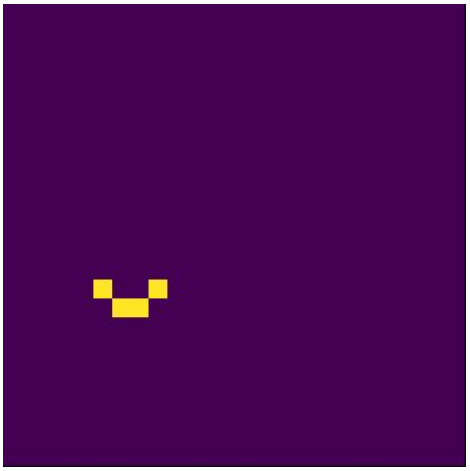
(run time was around 0.17 sec)

running the code for 568 iterations with a *size_of_board=100, starting_position=4 and rules=B3/S23 will give us:*



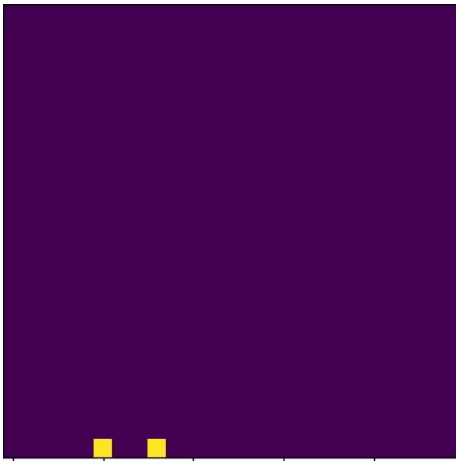
(run time was around 108 sec)

running the code for 10 iterations with a *size_of_board=25, starting_position=6 and rules=B2/S0 will give us:*



(run time was around 0.11 sec)

running the code for 50 iterations with a *size_of_board=25, starting_position=6 and rules=B2/S0 will give us:*

 *(run time was around 0.6 sec)*

Remarks:

- **An interface was uploaded to Moodle**. You should be familiar with it and use it while implementing your solution. You can start coding with our template file that was uploaded to Moodle.
- Your code will be uploaded using an import statement into our auto-grading code, therefore you should not run any code outside of the class member that you will create. For debugging purposes **you should use the statement if __name__ == '__main__'**.You can start coding with our template file that was uploaded to Moodle.
- **The board game is always cubicle and is not periodical space**. Meaning a cell on the borders have only five (or three) neighbors.
- When asking for a starting position in the middle of the board, assume the board size will be odd.
- Document your code - for yourself, and for good practice. Your comments should be written in **English only.**
- You should submit a **single .py** file through the Moodle website with the name of YOUR_ID_NUMBER.py were YOUR_ID_NUMBER should be your id number.
- The run time is limited to 20 minutes.
- Avoid copying!

References:

- https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
- http://www.conwaylife.com/wiki/Main_Page
- http://conwaylife.appspot.com/
- http://www.conwaylife.com/