**Introduction:**

Game of Life is a cellular automation imagined by John Conway in the 70s, and it is one of the best of all cellular automata. The rules are simple.  
This game is all about mathematical sense game, and it is a zero game player.

This games takes place on a 2D grid, and the cells could have two states, alive or dead.

**Rules:**

1. *A dead cell with exactly three living neighbours becomes alive.*
2. *A living cell with two or three living neighbours remains alive.*
3. *In all other cases, the cell becomes (or remains) dead.*

**Example:**

Let’s assume a simple initial state where only 3 cells are alive, *left cell*, *middle cell* and *right cell*.

The cells one the left and right have only one neighbour, the middle one, so they die. The middle cell has two neighbours, left and right so it stays alive. The top and bottom cells have 3 neighbours so they become alive. In this case, the game enters in an infinite loop, this is called *blinker*.

We can have multiple stable structures like *blocks* and *beehives*.

The pattern I am using in my code is *the glider cannon*.

**Python Code:**

As long as this game is in 2D, I used the library *numpy* and *pygame*.

I created 3 functions, *update*, *init* and *main*.

The variables are initialized in the *init* function. In this function, we create small patterns called *gliders*, which move in a straight line away from the *glider gun*. The state is saved in a 2D field. The pattern of the glider is saved in the variable named *pattern* and copied to the top left corner of the field containing the initial state of the simulation. Finally, the *cell* is returned.

The *update* function is the core of the simulation. This function calculates the status of the cells in the next time step and applies the rules of the game of life to all cells. At the beginning of the function, a new 2D field called *next* is created and initialized with zeros. The new state of the cells is saved in this field later on so that it can be returned at the end of the function. The function then iterates over all the cells in a loop. The number of living neighbouring cells is determined for each cell. Then the rules are applied.

Since the field *next* is already initialized with zeros, dead cells are not written into it. Finally, each cell is assigned a color that corresponds to its status. After the calculation is complete, the new state of the simulation is returned. The time step calculation is complete.

The *main* loop of the program is short. It initializes the program and contains an event loop with which *pygame* checks whether the program has been ended by the user. As long as the program has not ended, the *update* function is called and the simulation continues.