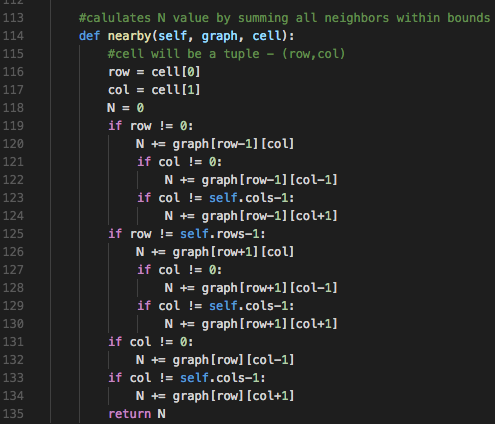
Our implementation of Conway’s game of life uses both Tkinter to animate the output and the generations are written to the file “outLife.txt”. We chose to write the file as a backup in case TKinter does not work and we found it easier to see how the game progresses through each generation in the file.

Our animation code mostly comes from the Tkinter example in the assignment 3 description. The speed can be toggled by changing the sleep value in line 42.

The IOGame function reads and parses the input from “inLife.txt”, the resulting graph and the number of generations are passed as arguments in the instantiation of a new Conway object.

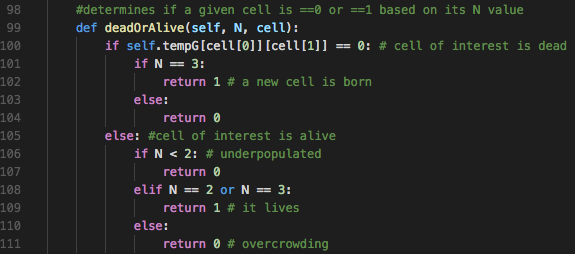
For each generation, the N values are calculated from the surrounding neighbours of each cell.

The N value is then used to determine whether or not the cell will become a 0 or a 1. After all of the N values are computed for each cell the graph is updated to reflect the new generation and the output is shown in the Tkinter window.



To calculate the N value for a given square we sum the surrounding 8 cells. We check that we are only summing cells within the bounds of the grid given the assumption that all cells outside of the graph are dead.

If a cell is dead and it has three alive neighbours, its value is set to 1. Likewise, if a cell is alive and has 2 or 3 neighbors its value stays as 1. All other cases lead to the cell dying. We implemented these rules in the deadOrAlive function and return either 0 or 1 based on the cell’s N value.



The two screenshots below show the sample input of the Gosper glider gun and part of the animated output from a generation of the game. Clicking the exit button before the generations are finished interrupts writing to the file.

