**Assignment 2 – Artificial Life**

**CISC 352**

**Due April 2, 2018**

**Spencer Edwards - 10129952**

**Hannah LeBlanc - 10091837**

**Conway’s Game of Life**

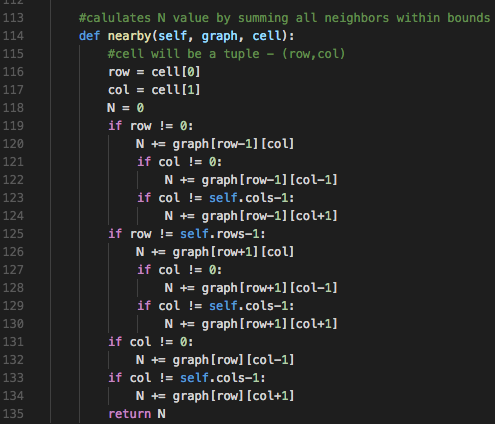
Our implementation of Conway’s game of life uses both Tkinter to animate the output and the generations are written to the file “outLife.txt”. We chose to write the file as a backup in case TKinter does not work and we found it easier to see how the game progresses through each generation in the file.

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.

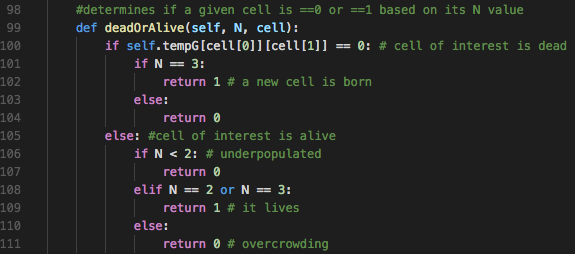
For each generation, the N values are calculated from the surrounding neighbours of each cell.

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.

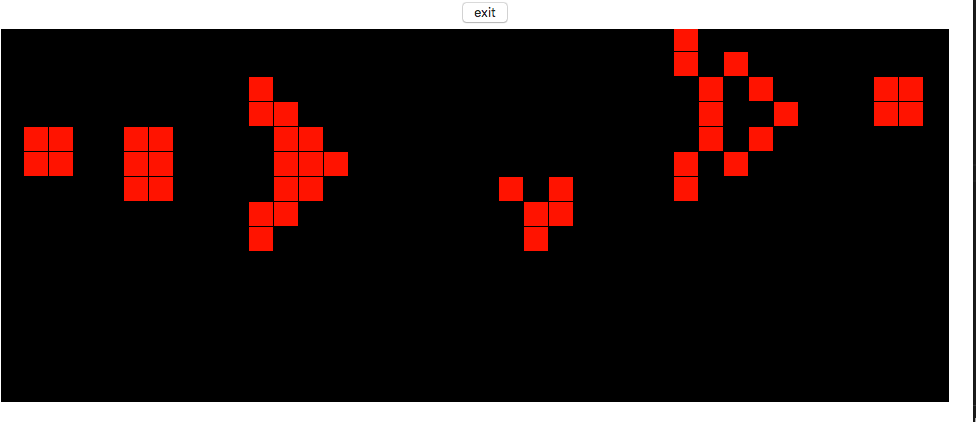
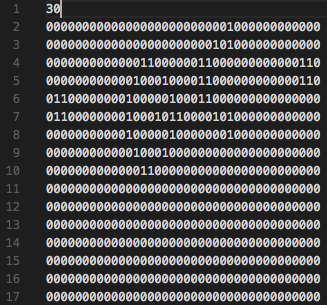


To calculate the N value for a given square we sum the surrounding 8 cells. We check that we are only summing cells within the bounds of the grid given the assumption that all cells outside of the graph are dead.

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.



**Craig Reynolds’ Boids System**

A Boids class has been implemented which represents one instance of the Craig Reynolds’ Boids System. The class has a variety of attributes which define the instance, ranging from information about the boids to constants for the 2D graphics. The graphics were implemented using the built-in Python tkinter library. This library isn’t great, but it is significantly faster than using Turtle, which is the original library we tried to use. The tkinter update and mainloop functions are used to handle the iterative nature of this artificial life demonstration, where one iteration is a single frame of the boids.

The boids are initialized to positions in a roughly equal distribution. This is done by the initializeBoids function, which divides the screen into quadrants and places one quarter of the boids in random positions in each quadrant. The velocity of each boid is then initialized towards the centre of the screen.

Next the runBoids function is called, which continuously calls drawBoids, then moveAllBoids, and then the tkinter update method calls runBoids again (waiting at least 40ms before calling itself again, though our laptops were to slow to handle this speed, perhaps the computer this will be tested on is faster).

moveAllBoids applies each of the rules to each boid, thereby handling moving all the boids from one frame to another. For the first 15 seconds, it only applies the 3 main rules: cohesion, separation, and alignment. Between 15 and 25 seconds a northern wind is applied, so the flock of boids is blown south (towards the bottom of the screen) during this time, and tends to remain here even once the wind has ceased (why would they move north again?). Another additional rule that is applied the entire time is bounding the position. We noticed that the flock would sometimes migrate off the screen and not return, so we implemented a feature which treats the screen as a semi-bounded area. Boids can still technically move off the screen, but if they are found to be off the screen a wall force is applied to the velocity of the boid to force the boid back onto the screen.

drawBoids refreshes the screen by deleting all the elements from it, and then draws each boid in the shape of an oval onto the screen in its respective spot.

A few helper functions were written as well to perform operations such as: vector addition, vector subtraction, Euclidean distance between two vectors, and the angle between to vectors/points on the screen (the last one was used to initialize the boid velocities towards the centre).

It should be noted that during development we had issues with the boids consistently flying north, which ended up being due to a switched around < sign. We asked the professor for help diagnosing this, and he also provided us with an alternative boids initialization function called random\_start. This function initializes boids along all 4 edges of the screen, so they naturally come towards each other in the centre. I have left our iniatlizeBoids function as the one in use. But to try his you can just switch the comment tags # in the class constructor. They both work quite well!