Assignment 3

Artificial Intelligence

CISC 352, Winter, 2019

Due Wednesday, April 3rd, 2019 before Midnight

Programming Language Requirement: Python 3

Max Group Size: 6

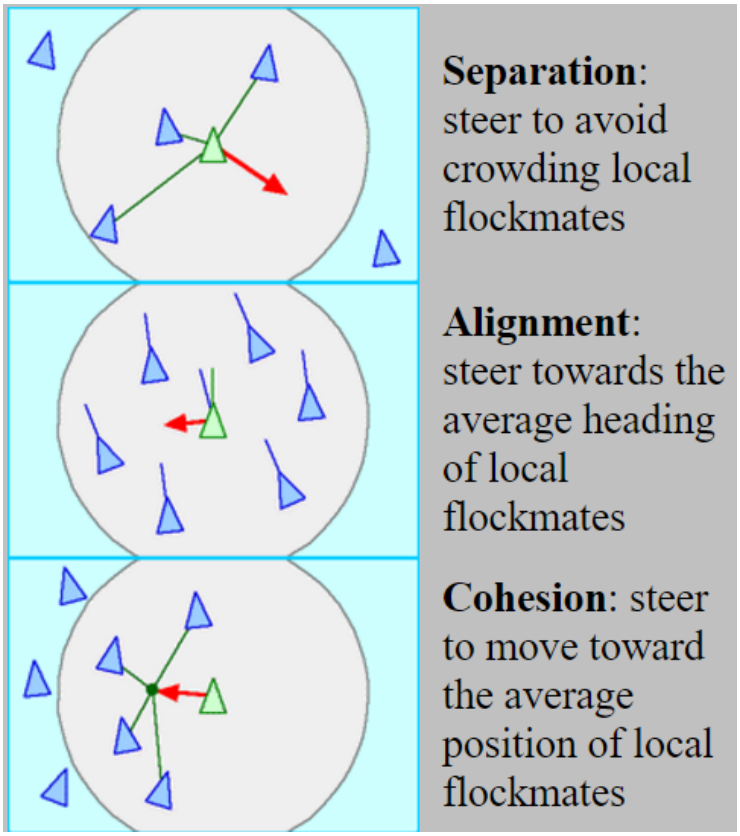
Preamble

Artificial Intelligence is about solving complex problems by mimicking the behavior of humans and animals. This term, you have solved several complex problems. Sometimes the difficulty of such problems may have overwhelmed you, but most of you have developed solutions, some elegant, some not, in the end. I hope through this journey of AI algorithms, you have learned a great deal, become better programmers, and gained some confidence when confronted with complex problems. With that in mind, I present you with the following final assignment.

Note that on this assignment, bonus points, up to a max score of 120, will be considered, based on factors described in the problem. If you go over 100 points, that's perfectly fine. The grade will be out of 100, so if you get 105, for instance, the bonus can help your previous scores and overall grade in the class.

Artificial Life: Craig Reynolds' Boids System

Grading: Works (40pts) / Does not Work (0pts)



Task: Simulate the Boids System. This requires a visual display. This system models coordinated animal motion such as flocks of birds and schools of fish. Rather than having an overall mechanism for controlling the flock, the system has just a few rules that each individual boid obeys. The basic flocking model consists of three simple steering behaviors which describe how an individual boid maneuvers based on the positions and velocities of its nearby flockmates.

Rule 1 (Cohesion): Boids tend to fly towards the center of mass of neighboring boids.

Rule 2 (Separation): Boids tend to keep a small distance away from other objects and boids.

Rule 3 (Alignment): Boids tend to match velocity with nearby boids.

Bonus: We went over several improvements to the basic algorithm, which can earn bonus points. First, simulate the basic system. Then, for example, if you implement wind, after ~10 seconds, start the wind going for a bit (perhaps another ~10 seconds) to show it working. And then simulate the next feature you implemented, such as perching, and so on. **Be sure to tell me in the Technical Document what you implemented, in what order they will be displayed, and about how long I should expect to wait to see it working.**

Program File Name: boids.py

Artificial Life: Conway's Game of Life

Grading: Works (40pts) / Does not Work (0pts)

Input: Implement Conway's Game of Life given an input configuration (m x n grid of 0's and 1's, where 4<=m<=100 and 4<=n<=100). The input will consist of an initial condition, as shown, along with the number of generations to simulate. There will only be one input configuration per file, as shown below: 3 00000 01110 00000 00000 00000	**Output:** The output does not require a 2D graphical display. If you do not use a graphical display, simply output the generations to the output file, as shown below: Generation 0 00000 01110 00000 00000 00000 Generation 1 00100 00100 00100 00000 00000 Generation 2 00000 01110 00000 00000 00000 Generation 3 00100 00100 00100 00000 00000

Bonus: Having a live graphical display earns **10** bonus points.

Program File Name: life.py

Input File Name: inLife.txt

Output File Name (for non-visual programs): outLife.txt

Technical Document

Grading: 20pts

Task: The third part is a grade on your technical document describing your algorithms. This is worth **20 points**. The grade will include grammar, spelling, presentation quality, and the description of your algorithms. I should be able to know how your algorithms work from reading the document. **Also, be sure to tell me in the Technical Document what you implemented in the Boids system, in what order they will be displayed, and about how long I should expect to wait to see it working.**

Regarding presentation quality, **do not submit a .txt file**. The document should be presented as nicely as possible, with headings where appropriate. **Put the name and student ids of all group members on this document. Convert the file to a PDF and name it assignment3.pdf.**

Grading

1. Craig Reynold's Boids System

   Works: 40 points

   Doesn't Work: 0 points

2. Conway's Game of Life

   Works: 40 points

   Doesn't Work: 0 points

3. Technical Document
   20 points

The total points you can earn for this assignment is **120**.

This assignment is **11%** of your total grade for the course.

Submit the program and technical document, one per group, compressed together in a zip file named **assignment3.zip** to the onQ Assignment 3 Dropbox.

Notes

```
PROCEDURE move_all_boids_to_new_positions()

        Vector v1, v2, v3
        Boid b

        FOR EACH BOID b
                v1 = rule1(b)
                v2 = rule2(b)
                v3 = rule3(b)

                b.velocity = b.velocity + v1 + v2 + v3
                b.position = b.position + b.velocity
        END

END PROCEDURE

PROCEDURE rule1(boid bJ)

        Vector pcJ

        FOR EACH BOID b
                IF b != bJ THEN
                        pcJ = pcJ + b.position
                END IF
        END

        pcJ = pcJ / N-1

        RETURN (pcJ - bJ.position) / 100

END PROCEDURE
```

Craig Reynold's Boids System

```
PROCEDURE rule2(boid bJ)

        Vector c = 0;

        FOR EACH BOID b
                IF b != bJ THEN
                        IF |b.position - bJ.position| < 100 THEN
                                c = c - (b.position - bJ.position)
                        END IF
                END IF
        END

        RETURN c

END PROCEDURE
```

```
PROCEDURE rule3(boid bJ)

        Vector pvJ

        FOR EACH BOID b
                IF b != bJ THEN
                        pvJ = pvJ + b.velocity
                END IF
        END

        pvJ = pvJ / N-1

        RETURN (pvJ - bJ.velocity) / 8

END PROCEDURE
```

Conway's Game of Life

The pattern must emerge from Conway's rules. Conway's game of life is described here:

A cell C is represented by a 1 when alive, or 0 when dead, in an m × m square neighborhood of cells.

Calculate N, the sum of live cells in C's eight-location neighborhood. Then, cell C is alive or dead in the next generation based on the following table:

C	N	New C
1	0 or 1	0, Died of Loneliness
1	4,5,6,7, or 8	0, Died of Overcrowding
1	2 or 3	1, Lives
0	3	1, It takes 3 to give birth!
0	0,1,2,4,5,6,7, or 8	0, Barren

Assume cells beyond the boundary are always dead.

The "game" is a zero-player game, meaning that its evolution is determined by its initial state, needing no input from human players. Interacting with the Game of Life involves creating an initial configuration and observing how it evolves.

See https://en.wikipedia.org/wiki/Conway's_Game_of_Life for some neat input patterns to test. Neat things here, too: http://web.stanford.edu/~cdebs/GameOfLife/

Python Graphics

There are many packages you can use in Python 3 for graphics. The following is an example of using tkinter with a partial solution to Conway's Game of Life. Note that the code will not work as-is. It contains a good amount of help to get started, though.

```
from tkinter import *

grid=[]
m = 25

def main():
    #MAIN
    get_in()
    initialise()
    mainloop()

def initialise():
    build_graph()

def build_graph():
    global graph
    global m
    WIDTH = m*len(grid[0])
    HEIGHT = m*len(grid)
    root = Tk()
    root.overrideredirect(True)
    root.geometry('%dx%d+%d+%d' % (WIDTH, HEIGHT, (root.winfo_screenwidth() - WIDTH) / 2, (root.winfo_screenheight() -
HEIGHT) / 2))
    root.bind_all('<Escape>', lambda event: event.widget.quit())
    graph = Canvas(root, width=WIDTH, height=HEIGHT, background='white')
    graph.after(40, update)
    graph.pack()

def update():
    draw()
    graph.after(500,update)

def draw():
    global m
    newGrid = next_gen()
    graph.delete(ALL)

    row = 0
    while row < len(newGrid):
        col = 0
        while col < len(newGrid[0]):
            cell = newGrid[row][col]
            startX = m*col
            endX = startX+m
            startY = m*row
            endY = startY+m
            if cell == 1:
                graph.create_rectangle(startX,startY,endX,endY,fill="red")
            else:
                graph.create_rectangle(startX,startY,endX,endY,fill="black")
            col = col+1
        row = row+1

    graph.update()
```