LANGTON'S ANT

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: JUÁREZ MARTÍNEZ GENARO

GRUPO: 3CM5

Noviembre 2019

Índice

1. INTRODUCTION

Langton's ant is a two-dimensional universal Turing machine with a very simple set of rules but complex emergent behavior. It was invented by Chris Langton in 1986 and runs on a square lattice of black and white cells. The universality of Langton's ant was proven in 2000. The idea has been generalized in several different ways, such as turmites which add more colors and more states.

1.1. Rules

Squares on a plane are colored variously either black or white. We arbitrarily identify one square as the "ant". The ant can travel in any of the four cardinal directions at each step it takes. The "ant" moves according to the rules below:

- At a white square, turn 90° right, flip the color of the square, move forward one unit
- At a black square, turn 90° left, flip the color of the square, move forward one unit

Langton's ant can also be described as a cellular automaton, where the grid is colored black or white and the "ant" square has one of eight different colors assigned to encode the combination of black/white state and the current direction of motion of the ant.

2. PROGRAM FUNCTIONALITY

When running the program the user can select in between different simulation options, these include:

- Console Dimension: This refers to the size in pixels of the console where the simulation will be displayed.
- How close ants should be: This means how conglomerated the ants will be, the greater the number the closer the range.
- Number of ants
- Queen life steps
- Soldier life steps
- Worker life steps
- Percentage of queens at start
- Percentage of soldiers at start
- Percentage of workers at start

By playing around with all these options one can find interesting behaviours in the colony.
The program also generates a density histogram of the ants over the space.

3. TESTING THE PROGRAM

When you run the program it looks like this:

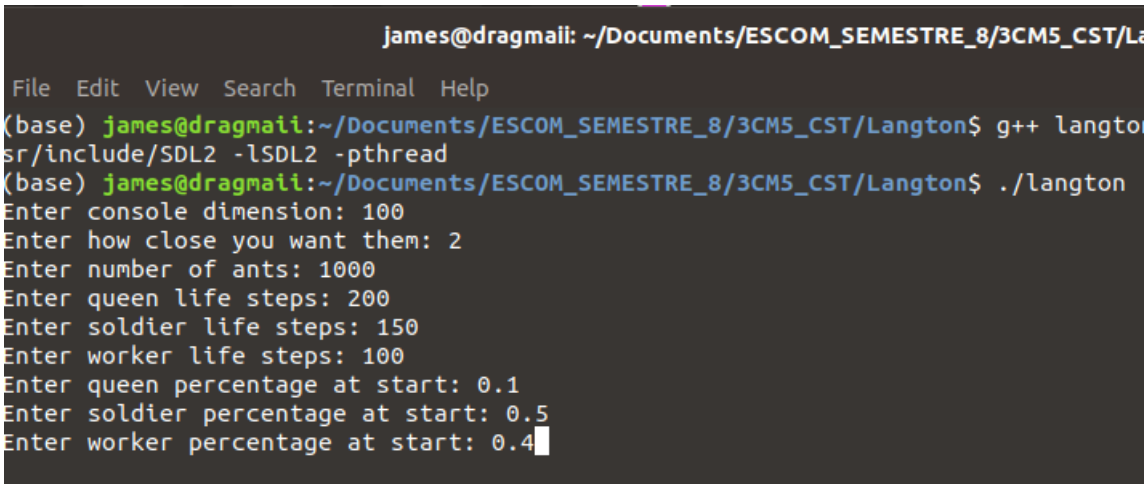

Figura 1: Program menu

Asking you to fulfill all the parameters. Then you will see the simulation running, showing in the console the number of iteration, workers population, soldiers population and queens population, respectively.

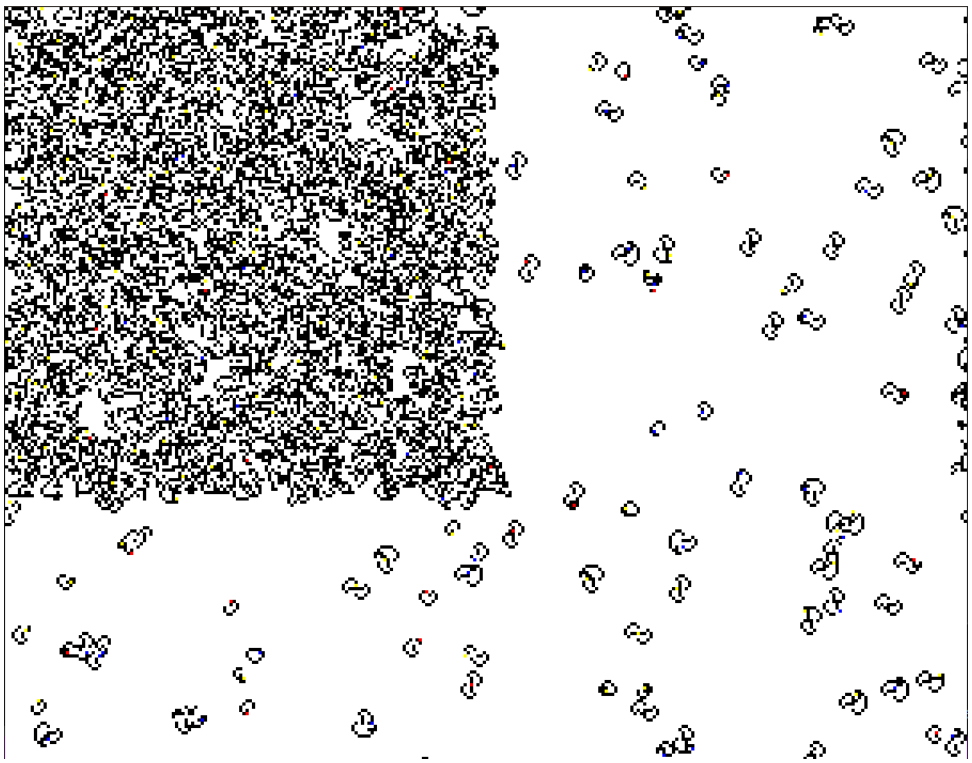

Figura 2: Simulation running 1

```
(base) james@dragmaii:~/Documents/ESCOM_SEMESTRE
Enter console dimension: 300
Enter how close you want them: 2
Enter number of ants: 1000
Enter queen life steps: 200
Enter soldier life steps: 150
Enter worker life steps: 100
Enter queen percentage at start: 0.1
Enter soldier percentage at start: 0.4
Enter worker percentage at start: 0.5
0 500 400 100
1 500 400 100
2 500 400 100
3 500 400 100
4 500 400 100
5 500 400 100
6 500 400 100
7 500 400 100
8 500 400 100
9 500 400 100
10 500 400 100
11 500 400 100
12 500 400 100
```

Figura 3: Population on console

Accompained by the graph of the density:

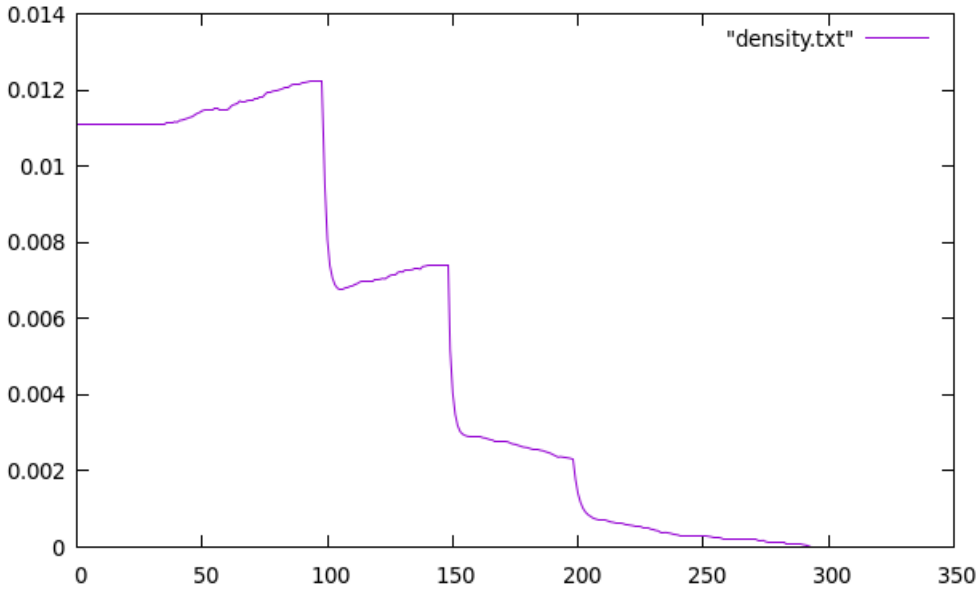

Figura 4: Density Graph

4. CODE

```
// V4
// Includes support for multiple class of ants (queen, soldier, worker) and life span steps
// Ants are divided in 3 vectors
// Supports reproduction up to 10000 soldiers and 1 queen in the same square
// Supports queen anhilitanion when in the same square
#include <iostream>
#include <vector>

using namespace std;

#include "olcConsoleGameEngineSDL.h"

class OneLoneCoder_GameOfLife : public olcConsoleGameEngine
{
public:
	int queen_life, soldier_life, worker_life;
	double queen_percentage, soldier_percentage, worker_percentage;
	int nants, close_range;

	OneLoneCoder_GameOfLife(int nants, int close_range, int queen_life, int soldier_life, int
	    worker_life, double queen_percentage, double soldier_percentage, double
	    worker_percentage)
	{
		m_sAppName = L"Langton's Ant James";
		this->nants = nants;
		this->close_range = close_range;
		this->queen_life = queen_life;
		this->soldier_life = soldier_life;
		this->worker_life = worker_life;
		this->queen_percentage = queen_percentage;
		this->soldier_percentage = soldier_percentage;
		this->worker_percentage = worker_percentage;
	}

private:
	int *m_output, *m_state;
	int i, j, k;
	vector<int *> worker_ants;
	vector<int *> soldier_ants;
	vector<int *> queen_ants;
	int *ant, *soldier_ant, *queen_ant;
	int *newborn_ants[1000];
	int queen_apogee_begin, queen_apogee_end, newborn_probability;
	int newborn_counter, queens_counter, queens_to_erase_cnt;
	bool reproduction180angle = true, reproduction90angle = true, allowreproduction;
	bool this_ant_erased, already_dead;
	int queens_to_erase[1000], iterations = 0;
	double average;
	ofstream density_writer;

protected:
	// Called by olcConsoleGameEngine
	virtual bool OnUserCreate()
	{
		int no_queens = nants * queen_percentage;
		int no_soldiers = nants * soldier_percentage;
		int no_workers = nants * worker_percentage;
		// int no_queens = 3, no_soldiers = 0, no_workers = 0;
		int cnt_soldiers = 0, cnt_workers = 0, cnt_queens = 0;
		queen_apogee_begin = (queen_life / 2) - ((queen_life - 10) / 3);
		queen_apogee_end = (queen_life / 2) + ((queen_life - 10) / 3);

		m_output = new int[ScreenWidth() * ScreenHeight()];
		m_state = new int[ScreenWidth() * ScreenHeight()];

		memset(m_output, 0, ScreenWidth() * ScreenHeight() * sizeof(int));
		memset(m_state, 0, ScreenWidth() * ScreenHeight() * sizeof(int));

		for(i = 0; i < nants; i++)
```

```
        {
                ant = (int *) malloc(4 * sizeof(int));
                if(cnt_queens < no_queens) // QUEEN
                {
                        ant[0] = rand() %(ScreenWidth() / close_range); // X position
                        ant[1] = rand() %(ScreenHeight() / close_range); // Y position
                        ant[2] = rand() %4;                     // Direction N-E-S-W
                        ant[3] = queen_life;                                    // Life
                            steps
                        cnt_queens++;
                        queen_ants.push_back(ant);
                }
                else if(cnt_soldiers < no_soldiers)
                {
                        ant[0] = rand() %(ScreenWidth() / close_range); // X position
                        ant[1] = rand() %(ScreenHeight() / close_range); // Y position
                        ant[2] = rand() %4;
                        ant[3] = soldier_life;
                        cnt_soldiers++;
                        soldier_ants.push_back(ant);
                }
                else if(cnt_workers < no_workers)
                {
                        ant[0] = rand() %(ScreenWidth() / close_range); // X position
                        ant[1] = rand() %(ScreenHeight() / close_range); // Y position
                        ant[2] = rand() %4;
                        ant[3] = worker_life;
                        cnt_workers++;
                        worker_ants.push_back(ant);
                }
        }
        //OPENING FILES
        density_writer.open("density.txt", ios_base::out | ios_base::trunc);
        if(!density_writer)
        {
                cout << "Error: Cannot open 'density' file" << endl;
                return true;
        }
        // SETTING PRECISION
        cout.precision(8);
        density_writer.precision(8);

        return true;
}

// Called by olcConsoleGameEngine
virtual bool OnUserUpdate(float fElapsedTime)
{
        cout << iterations++ << " " << worker_ants.size() << " " << soldier_ants.size() <<
            " " << queen_ants.size() << endl;

        if(m_keys[VK_SPACE].bHeld)
                return true;

        this_thread::sleep_for(100ms);

        auto cell = [&](int x, int y)
        {
                return m_output[y * ScreenWidth() + x];
        };

        auto setcell = [&](int x, int y)
        {
                m_state[y * ScreenWidth() + x] = 1;
        };

        auto unsetcell = [&](int x, int y)
        {
                m_state[y * ScreenWidth() + x] = 0;
        };

        // Store output state
```

```
for(i = 0; i < ScreenWidth()*ScreenHeight(); i++)
        m_output[i] = m_state[i];

for(int x = 0; x < ScreenWidth(); x++)
        for (int y = 0; y < ScreenHeight(); y++)
        {
                if(cell(x, y))
                        Draw(x, y, PIXEL_SOLID, FG_BLACK);
                else
                        Draw(x, y, PIXEL_SOLID, FG_WHITE);

        }

// WORKER ANTS
i = 0;
for(auto iterator = worker_ants.begin(); i < worker_ants.size(); iterator++, i++)
{
        ant = *iterator;
        // DRAWING WORKER ANT
        Draw(ant[0], ant[1], PIXEL_SOLID, FG_RED);
        // ANT COMPUTATION
        if(cell(ant[0], ant[1]) == 0)
        {
                if(ant[2] == 3)
                        ant[2] = 0;
                else
                        ant[2]++;
                setcell(ant[0], ant[1]);
        }
        else
        {
                if(ant[2] == 0)
                        ant[2] = 3;
                else
                        ant[2]--;
                unsetcell(ant[0], ant[1]);
        }
        // MOVING ANT
        if(ant[2] == 0) // Looking North
        {
                if(ant[1] == 0)
                        ant[1] = ScreenHeight() - 1;
                else
                        ant[1]--;
        }
        else if(ant[2] == 1) // Looking East
        {
                if(ant[0] == (ScreenWidth() - 1))
                        ant[0] = 0;
                else
                        ant[0]++;
        }
        else if(ant[2] == 2) // Looking South
        {
                if(ant[1] == (ScreenHeight() - 1))
                        ant[1] = 0;
                else
                        ant[1]++;
        }
        else if(ant[2] == 3) // Looking West
        {
                if(ant[0] == 0)
                        ant[0] = ScreenWidth() - 1;
                else
                        ant[0]--;
        }
        // UPDATING LIFE STEPS
        ant[3]--;
        // ERASING ANT WHEN LIFE STEPS REACH 0
        if(ant[3] == 0)
                worker_ants.erase(iterator);
}
```

```
// SOLDIER ANTS
i = 0;
for(auto iterator = soldier_ants.begin(); i < soldier_ants.size(); iterator++, i++)
{
        ant = *iterator;
        // cout << "Soldier" << " " << ant[0] << " : " << ant[1] << " : " << ant[2]
             << " : " << ant[3] << endl;

        // DRAWING SOLDIER ANT
        Draw(ant[0], ant[1], PIXEL_SOLID, FG_BLUE);
        // ANT COMPUTATION
        if(cell(ant[0], ant[1]) == 0)
        {
                if(ant[2] == 3)
                        ant[2] = 0;
                else
                        ant[2]++;
                setcell(ant[0], ant[1]);
        }
        else
        {
                if(ant[2] == 0)
                        ant[2] = 3;
                else
                        ant[2]--;
                unsetcell(ant[0], ant[1]);
        }
        // MOVING ANT
        if(ant[2] == 0) // Looking North
        {
                if(ant[1] == 0)
                        ant[1] = ScreenHeight() - 1;
                else
                        ant[1]--;
        }
        else if(ant[2] == 1) // Looking East
        {
                if(ant[0] == (ScreenWidth() - 1))
                        ant[0] = 0;
                else
                        ant[0]++;
        }
        else if(ant[2] == 2) // Looking South
        {
                if(ant[1] == (ScreenHeight() - 1))
                        ant[1] = 0;
                else
                        ant[1]++;
        }
        else if(ant[2] == 3) // Looking West
        {
                if(ant[0] == 0)
                        ant[0] = ScreenWidth() - 1;
                else
                        ant[0]--;
        }
        // UPDATING LIFE STEPS
        ant[3]--;
        // ERASING ANT WHEN LIFE STEPS REACH 0
        if(ant[3] == 0)
                soldier_ants.erase(iterator);
}

// FINDING QUEEN ANTS IN THE SAME POSITION
i = 0;
queens_to_erase_cnt = 0;
for(auto iterator = queen_ants.begin(); iterator != queen_ants.end(); iterator++, i
    ++)
{
        this_ant_erased = false;
        already_dead = false;
```

```
for(j = 0; j < queens_to_erase_cnt; j++)
{
        if(i == queens_to_erase[j])
                already_dead = true;
}
if(already_dead)
        continue;

ant = *iterator;
// LOOKING FOR ANY OTHER QUEEN IN THE SAME POSITION
queens_counter = 0;
for(auto queen_iterator = queen_ants.begin(); queen_iterator != queen_ants.
    end(); queen_iterator++, queens_counter++)
{
        already_dead = false;
        for(j = 0; j < queens_to_erase_cnt; j++)
        {
                if(queens_counter == queens_to_erase[j])
                        already_dead = true;
        }
        if(already_dead)
                continue;

        queen_ant = *queen_iterator;

        // cout << "Ant " << i << " vs " << queens_counter << endl;
        if(iterator != queen_iterator && ant[0] == queen_ant[0] && ant[1]
            == queen_ant[1])       // A QUEEN IN THE SAME POSITION NOT BEING
             ITSELF
        {
                if(ant[3] < queen_apogee_begin || ant[3] > queen_apogee_end
                    )    // THIS QUEEN IS TOO YOUNG OR TOO OLD
                {
                        if(queen_ant[3] >= queen_apogee_begin && queen_ant
                            [3] <= queen_apogee_end)       // THE OTHER
                            QUEEN IS IN APOGEE
                        {
                                // cout << i << " eliminated by " <<
                                    queens_counter << endl;
                                // this_thread::sleep_for(1000ms);
                                queens_to_erase[queens_to_erase_cnt++] = i;
                                this_ant_erased = true;
                        }
                        else if((rand() % 2) == 0)       // BOTH ARE TOO
                            YOUNG OR TOO OLD

                        {
                                queens_to_erase[queens_to_erase_cnt++] = i;
                                this_ant_erased = true;
                        }
                        else
                                queens_to_erase[queens_to_erase_cnt++] =
                                    queens_counter;
                }
                else // THIS QUEEN IS IN ITS APOGEE
                {
                        if(queen_ant[3] < queen_apogee_begin || queen_ant
                            [3] > queen_apogee_end)        // THE OTHER
                            QUEEN IS TOO YOUNG OR TOO OLD
                        {
                                queens_to_erase[queens_to_erase_cnt++] =
                                    queens_counter;
                        }
                        else if((rand() % 2) == 0)       // BOTH ARE IN
                            THEIR APOGEE
                        {
                                queens_to_erase[queens_to_erase_cnt++] = i;
                                this_ant_erased = true;
                        }
                        else
                                queens_to_erase[queens_to_erase_cnt++] =
                                    queens_counter;
```

```
                        }
                }
                if(this_ant_erased)
                {
                        // cout << "This ant erased" << endl;
                        break;
                }
        }
}
// cout << "FINISHED" << endl;
// ACTUALLY ERASING QUEEN ANTS
i = 0;
j = 0;
for(auto iterator = queen_ants.begin(); j < queens_to_erase_cnt; i++)
{
        // cout << i << " == " << queens_to_erase[j] << endl;
        if(i == queens_to_erase[j])
        {
                // cout << "Erasing " << i << endl;
                queen_ants.erase(iterator);
                if(j != queens_to_erase_cnt)
                        iterator++;
                j++;
        }
}
// cout << "KILLED" << endl;
// QUEEN ANTS
i = 0;
queens_to_erase_cnt = 0;
newborn_counter = 0;
for(auto iterator = queen_ants.begin(); iterator != queen_ants.end(); iterator++, i
    ++)
{
        ant = *iterator;
        // cout << "Queen" << i << " " << ant[0] << " : " << ant[1] << " : " << ant
            [2] << " : " << ant[3] << endl;

        // DRAWING QUEEN ANTS
        Draw(ant[0], ant[1], PIXEL_SOLID, FG_YELLOW);

        // ANT COMPUTATION
        if(cell(ant[0], ant[1]) == 0)
        {
                if(ant[2] == 3)
                        ant[2] = 0;
                else
                        ant[2]++;
                setcell(ant[0], ant[1]);
        }
        else
        {
                if(ant[2] == 0)
                        ant[2] = 3;
                else
                        ant[2]--;
                unsetcell(ant[0], ant[1]);
        }
        // MOVING ANT
        if(ant[2] == 0) // Looking North
        {
                if(ant[1] == 0)
                        ant[1] = ScreenHeight() - 1;
                else
                        ant[1]--;
        }
        else if(ant[2] == 1) // Looking East
        {
                if(ant[0] == (ScreenWidth() - 1))
                        ant[0] = 0;
                else
                        ant[0]++;
        }
```

```
else if(ant[2] == 2) // Looking South
{
        if(ant[1] == (ScreenHeight() - 1))
                ant[1] = 0;
        else
                ant[1]++;
}
else if(ant[2] == 3) // Looking West
{
        if(ant[0] == 0)
                ant[0] = ScreenWidth() - 1;
        else
                ant[0]--;
}
// UPDATING LIFE STEPS
ant[3]--;
// ERASING ANT WHEN LIFE STEPS REACH 0
if(ant[3] <= 0)
{
        queens_to_erase[queens_to_erase_cnt++] = i++;
        continue;
}
// REPRODUCTION PERIOD
if(ant[3] >= queen_apogee_begin && ant[3] <= queen_apogee_end)
{
        // cout << "Reproduction Period" << endl;
        // LOOKING FOR ANY OTHER SOLDIER ANT IN THE SAME POSITION
        j = 0;
        for(auto soldier_iterator = soldier_ants.begin(); j < soldier_ants.
            size(); soldier_iterator++, j++)
        {
                soldier_ant = *soldier_iterator;

                // A SOLDIER IN THE SAME POSITION
                if(soldier_ant[0] == ant[0] && soldier_ant[1] == ant[1])
                {
                        allowreproduction = false;
                        if(((soldier_ant[2] + 2) % 4) == ant[2] &&
                            reproduction180angle)          // SOLDIER AND
                            QUEEN MET AT 180
                                allowreproduction = true;
                        else if(((soldier_ant[2] + 3) % 4) == ant[2] &&
                            reproduction90angle)     // SOLDIER AND QUEEN
                            MET AT 90
                                allowreproduction = true;

                        if(allowreproduction)
                        {
                                for(k = 0; k < 2; k++) // GIVING BIRTH TO
                                    TWO OTHER ANTS
                                {
                                        newborn_ants[newborn_counter] = (
                                            int *) malloc(4 * sizeof(int));
                                        // newborn_ants[newborn_counter][0]
                                             = ant[0] - (k + 1); // X
                                            position
                                        // newborn_ants[newborn_counter][1]
                                             = ant[1] - (k + 1); // Y
                                            position
                                        newborn_ants[newborn_counter][0] =
                                            rand() % ScreenWidth(); // X
                                            position
                                        newborn_ants[newborn_counter][1] =
                                            rand() % ScreenHeight(); // Y
                                            position
                                        newborn_ants[newborn_counter][2] =
                                            rand() % 4;
                                                        // Direction N-
                                            E-S-W
                                        newborn_counter++;
                                }
                        }
```

```
                                        }
                                }
                        }
                }
                // INSERTING NEW BORNS WHERE THEY BELONG
                for(k = 0; k < newborn_counter; k++)
                {
                        newborn_probability = rand() %100;
                        if(newborn_probability < 30)
                        {
                                newborn_ants[k][3] = worker_life;
                                worker_ants.push_back(newborn_ants[k]);
                        }
                        else if(newborn_probability < 60)
                        {
                                newborn_ants[k][3] = soldier_life;
                                soldier_ants.push_back(newborn_ants[k]);

                        }
                        else if(newborn_probability < 100)
                        {
                                newborn_ants[k][3] = queen_life;
                                queen_ants.push_back(newborn_ants[k]);
                        }
                }
                // ACTUALLY ERASING QUEEN ANTS
                i = 0;
                j = 0;
                for(auto iterator = queen_ants.begin(); i < queen_ants.size() && j <
                    queens_to_erase_cnt; iterator++, i++)
                {
                        if(i == queens_to_erase[j])
                        {
                                queen_ants.erase(iterator);
                                j++;
                        }
                }
                // WRITING
                density_writer << (double) (worker_ants.size() + soldier_ants.size() + queen_ants.
                    size()) / (ScreenWidth() * ScreenHeight()) << endl;

                return true;
        }
};


int main()
{
        // Seed random number generator
        srand(clock());

        int queen_life, worker_life, soldier_life;
        double queen_percentage, soldier_percentage, worker_percentage;
        int nants, console_dimension, close_range;
        string life_steps;

        cout << "Enter console dimension: ";
        cin >> console_dimension;
        cout << "Enter how close you want them: ";
        cin >> close_range;
        cout << "Enter number of ants: ";
        cin >> nants;
        cout << "Enter queen life steps: ";
        cin >> queen_life;
        cout << "Enter soldier life steps: ";
        cin >> soldier_life;
        cout << "Enter worker life steps: ";
        cin >> worker_life;
        cout << "Enter queen percentage at start: ";
        cin >> queen_percentage;
        cout << "Enter soldier percentage at start: ";
        cin >> soldier_percentage;
```

```
        cout << "Enter worker percentage at start: ";
        cin >> worker_percentage;

        // Use olcConsoleGameEngine derived app
        OneLoneCoder_GameOfLife game(nants, close_range, queen_life, soldier_life, worker_life,
            queen_percentage, soldier_percentage, worker_percentage);
        game.ConstructConsole(console_dimension, console_dimension, 3, 3);
        game.Start();

        return 0;
}
```