Langton's Ant: Algorithm implementation and Experiment

Vicky Huiqi ZHENG, Dewei ZOU

May 27, 2021

Contents

1 Introduction

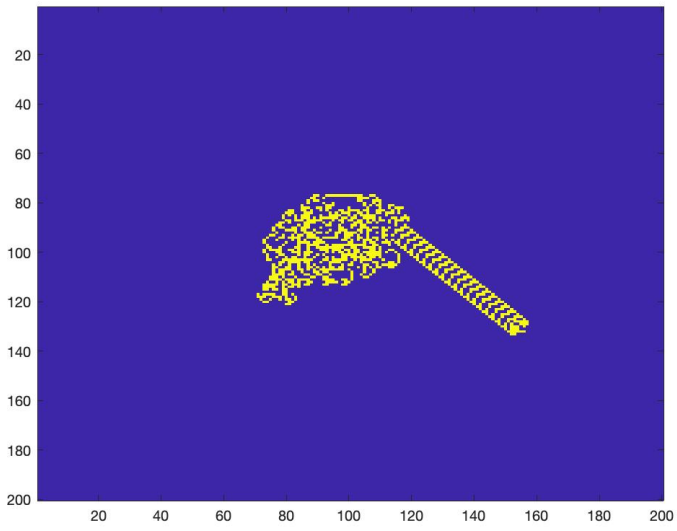
Langton´s Ant is named after his inventor Chris Langton in 1986 [1]. Basically, an ant is placed on the center of a two dimensional grid. The grid consists of squares, which can be colored white or black, followed by a initial set of rules. After about 11 000 steps, the ant produces a regular periodic pattern, which is called as an "highway". Since Langton´s Ant has an interesting and complex behavior so it is studied in several context. Later on, a generalization of the idea of Langton´s Ant has been occured, where people change the initial set of rules, like adding more colors or adding multiple ants.

Rules

At the beginning, there is an grid of white squares, where an ant is on the middle of the grid. The ant will follow a set of rules, where squares are colored either black or white. For each step the ant can move in four directions north, east, south and west and follows to the rules below:

- At a white square, turns to the right, the color of the actual square will be inversed, moves to the next square
- At a black square, turns to the left, the color of the actual square will be inversed, takes a step forward to the next square

We can also called it as a cellular automaton.



Langton's ant after 11,000 steps

2 Overview

In section 3, we will talk about the code in Java and matlab.
In section 4, we will talk about the result of our experiment:

- 2 ants with different position
- muliple ants with same starting point, but different direction
- ants that build a rectangle
- relationship of number of ants and highway

3 Basic Algorithm

3.1 In Matlab

We firstly use Matlab to implement the algorithm. Since some features are very easy to use.
And the code is:

```
height = 200;
width = 200;
step = 15000;
direction = [1,2,3,4]; % direction of the ant 1:N 2:E 3:S 4:W
ant = zeros(height,width); % black = 1, white = 0
x = height/2;
y = width/2; % middle of filed
d1 = 1;% initialize the direction
d2 = 1;
n=2;
for a= 1:n
    eval(['x' num2str(a) '=x']);
    eval(['y' num2str(a) '=y']);
end

for i=1:step
    for j= 1:n
        xt=eval(['x' num2str(j)]);
        yt=eval(['y' num2str(j)]);
        dt=eval(['d' num2str(j)]);
        [xt,yt] = coordinate(xt,yt,dt);
        if ant(xt,yt)==0
            ant(xt,yt) = 1;
            dt = turnLeft(dt); % turn left if white
        else
            ant(xt,yt) = 0;
            dt = turnRight(dt); % turn right if black
        end
        eval(['x' num2str(j) '=xt']);
        eval(['y' num2str(j) '=yt']);
        eval(['d' num2str(j) '=dt']);
     end
     if mod(i,1000)==0
        imagesc(ant);% output the image
        title("Steps :",i);
        pause(0.1);
    end
end

```

```
function [x,y] = coordinate(x0,y0,direction)
if direction==1
x = x0-1;
y = y0;
elseif direction==2
x = x0;
y = y0+1;
elseif direction==3
y = y0;
x = x0+1;
elseif direction==4
x = x0;
y = y0-1;
else
    error("Direction error");
end
end


function d = turnLeft(CurrentDirection) % d is the direction after turn right
if CurrentDirection==1
d = 4;
elseif CurrentDirection==2
    d = 1;
elseif CurrentDirection==3
    d =2;
elseif CurrentDirection==4
    d =3;
else
d=0;
    error("Direction error");
end
end

function d = turnRight(CurrentDirection) % d is the direction after turn right
if CurrentDirection==1
    d = 2;
elseif CurrentDirection==2
d = 3;
elseif CurrentDirection==3
    d =4;
elseif CurrentDirection==4
    d =1;
else
d=0;
error("Direction error");
end
end
```

Where $(x1, y1), (x2, y2)$ used to set the amount of the ants.

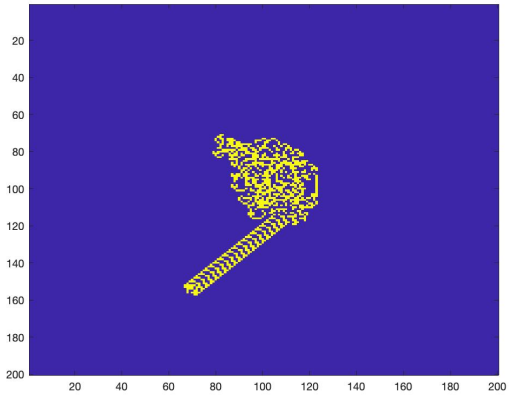
The idea for this code is simple and basic. We firstly create a *null matrix*:

$$\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

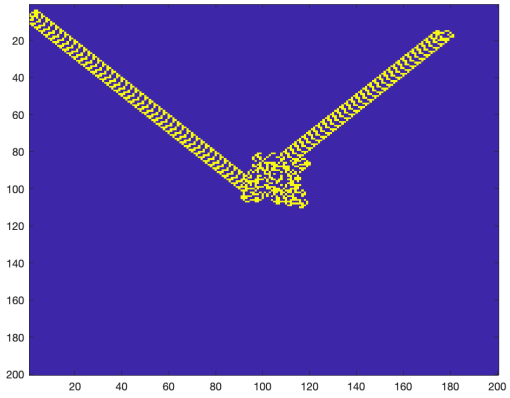
And think this as a coordinate system with x, y, therefore we set the ant at the middle, which is exactly what $\frac{1}{2}$ *width*, and $\frac{1}{2}$ *height* do. By following the rule of the ant's behaviour, if the ant steps on 0, then we turn it to 1. Otherwise turn it to 0.

Matlab's plotting feature is very powerful to this project, with simply by using **image()**, we would have the image as we expected.

The figure below shows the Langton´s Ant facing East, after about 11 000 steps.



And here we have the figure if we set the amount of ants as 2, and set the initial direction as *North*:



In next section we used the algorithm created by Sylvain Saurel [2].

3.2 In Java

First at all, a file with class **Ant** contains the basical statue of the ant:

```
public class Ant {

    public int x, y, steps;
    public Direction direction;

    public Ant(int x, int y, Direction direction) {
        this.x = x;
        this.y = y;
        this.direction = direction;
    }

    public boolean isInWorld(World w) {
    return 0 <= x && x < w.size && 0 <= y && y < w.size;
    }
    public void step(World w) {
        World.Color c = w.getCellColor(x, y);
        // we change direction according to the rules of the Langton'Ant
        direction = (World.Color.WHITE == c) ? direction.right() : direction.left();
        w.setCellColor(x, y, c.inverse()); // we inverse color
        x += direction.deltaX; // we update new coordinates for the ant
        y += direction.deltaY;
        steps++; // we increment steps
    }

public enum Direction {
        NORTH(0, -1), //ordinal=0
        EAST(1, 0),
        SOUTH(0, 1),
        WEST(-1, 0);

        public final int deltaX, deltaY;

        private Direction(int dX, int dY) {
            deltaX = dX;
            deltaY = dY;
        }

        // we return the Direction that is 90 degrees to the left
        public Direction left() {
            return Direction.values()[(this.ordinal() + 3) % 4];
        }

        // we return the Direction that is 90 degrees to the right
        public Direction right() {
            return Direction.values()[(this.ordinal() + 1) % 4];
        }
    }
}
```

In which **values()** method converts enum type to an array, and **ordinal()** returns the

order of an enum instance.

Here is the file for the grid for ant to move :

```
public class World {

    public static enum Color {
        WHITE, BLACK,GREEN;

        public Color inverse() {
            return WHITE.equals(this) ? BLACK : WHITE;
        }
    }

    private Color[][] cells;
    public int size;

    public World(int size) {
        this.size = size;
        cells = new Color[size][size];

        // we init the cells with white color
        for (int x = 0; x < size; x++) {
            for (int y = 0; y < size; y++) {
                cells[x][y] = Color.WHITE;
            }
        }
    }

    public Color getCellColor(int x, int y) {
        return cells[x][y];
    }

    public void setCellColor(int x, int y, Color c) {
        cells[x][y] = c;
    }

    public boolean checkHighwaySouth(int x, int y) {
        boolean highway=false;
        if (cells[x][y] == World.Color.BLACK && cells[x + 1][y + 1] == World.Color.
            BLACK && cells[x + 2][y + 2] == World.Color.BLACK && cells[x + 3][y + 3] ==
             World.Color.BLACK
              && cells[x][y + 1] == World.Color.BLACK && cells[x - 1][y + 1] == World.
                  Color.BLACK && cells[x - 2][y + 2] == World.Color.BLACK && cells[x -
                   3][y + 2] == World.Color.BLACK
              && cells[x - 3][y + 3] == World.Color.BLACK && cells[x - 3][y + 4] ==
                  World.Color.BLACK && cells[x - 4][y + 4] == World.Color.BLACK &&
                  cells[x - 4][y + 5] == World.Color.BLACK
              && cells[x - 4][y + 6] == World.Color.BLACK && cells[x - 5][y + 5] ==
                  World.Color.BLACK && cells[x - 4][y + 6] == World.Color.BLACK &&
                  cells[x - 3][y + 7] == World.Color.BLACK
              && cells[x - 2][y + 8] == World.Color.BLACK && cells[x - 2][y + 7] ==
                  World.Color.BLACK && cells[x - 2][y + 6] == World.Color.BLACK &&
                  cells[x - 1][y + 6] == World.Color.BLACK
              && cells[x - 1][y + 5] == World.Color.BLACK && cells[x - 1][y + 4] ==
                  World.Color.BLACK && cells[x][y + 4] == World.Color.BLACK && cells[x
                   + 1][y + 3] == World.Color.BLACK
```

```
42                && cells[x + 2][y + 3] == World.Color.BLACK) {
43            highway = true;
44        }
45        return highway;
46    }
47
48    public boolean checkHighwayNorth(int x, int y) {
49        boolean state = false;
50        if (cells[x][y] == World.Color.BLACK && cells[x][y - 1] == World.Color.BLACK &&
              cells[x][y - 2] == World.Color.BLACK && cells[x + 1][y - 2] == World.Color
              .BLACK
51                && cells[x + 1][y - 3] == World.Color.BLACK && cells[x + 1][y - 4] ==
                      World.Color.BLACK && cells[x + 2][y - 3] == World.Color.BLACK &&
                      cells[x][y - 5] == World.Color.BLACK
52                && cells[x - 1][y - 5] == World.Color.BLACK && cells[x - 1][y - 6] ==
                      World.Color.BLACK && cells[x - 1][y - 4] == World.Color.BLACK &&
                      cells[x - 2][y - 4] == World.Color.BLACK
53                && cells[x - 2][y - 4] == World.Color.BLACK && cells[x - 2][y - 3] ==
                      World.Color.BLACK && cells[x - 2][y - 2] == World.Color.BLACK &&
                      cells[x - 3][y - 2] == World.Color.BLACK
54                && cells[x - 4][y - 1] == World.Color.BLACK && cells[x - 5][y - 1] ==
                      World.Color.BLACK && cells[x - 6][y - 1] == World.Color.BLACK &&
                      cells[x - 5][y] == World.Color.BLACK
55                && cells[x - 1][y] == World.Color.BLACK && cells[x - 2][y + 1] == World.
                      Color.BLACK && cells[x - 3][y + 1] == World.Color.BLACK && cells[x -
                       4][y + 1] == World.Color.BLACK
56                && cells[x - 3][y + 2] == World.Color.BLACK) {
57            state = true;
58        }
59        return state;
60    }
61
62    public boolean checkHighwayEast(int x, int y) {
63        boolean state = false;
64        if (cells[x][y] == World.Color.BLACK && cells[x][y - 1] == World.Color.BLACK &&
              cells[x-1][y - 2] == World.Color.BLACK && cells[x - 1][y - 3] == World.
              Color.BLACK
65                && cells[x - 1][y - 4] == World.Color.BLACK && cells[x -2][y - 3] ==
                      World.Color.BLACK && cells[x -3][y - 2] == World.Color.BLACK &&
                      cells[x-3][y - 1] == World.Color.BLACK
66                && cells[x - 4][y - 1] == World.Color.BLACK && cells[x - 3][y] == World.
                      Color.BLACK && cells[x - 2][y +1] == World.Color.BLACK && cells[x -
                      2][y +2] == World.Color.BLACK
67                && cells[x - 1][y +2] == World.Color.BLACK && cells[x][y +2] == World.
                      Color.BLACK && cells[x][y +3] == World.Color.BLACK && cells[x +1][y
                      +3] == World.Color.BLACK
68                && cells[x +1][y +4] == World.Color.BLACK && cells[x +2][y+3] == World.
                      Color.BLACK && cells[x +3][y+2] == World.Color.BLACK && cells[x+2][y
                      +1] == World.Color.BLACK
69                && cells[x +3][y+1] == World.Color.BLACK && cells[x +4][y + 1] == World.
                      Color.BLACK && cells[x +1][y] == World.Color.BLACK && cells[x+2][y]
                      == World.Color.BLACK){
70            state = true;
71        }
72        return state;
73    }
74}
```

inverse() function changes the color while the ant steps on.

The coordinate x and y are represended by using array, which apparently is a $2-$ *dimensional* plane. And then set the size of the array to create a grid for ant to move.

In next file, there is the **main()** fuction, and imported method from *javax.swing* for the UI:

```
1
2  import java.awt.Color;
3  import java.awt.Dimension;
4  import java.awt.FlowLayout;
5  import java.awt.Graphics;
6  import java.util.Timer;
7  import java.util.TimerTask;
8
9  import javax.swing.JFrame;
10 import javax.swing.JLabel;
11 import javax.swing.JPanel;
12 import javax.swing.SwingUtilities;
13
14 public class LangtonAnt {
15
16     // we use Swing for the UI
17     private JFrame frame;
18     private JPanel antPanel;
19     private World world;
20     private Ant ant;
21
22     public class AntPanel extends JPanel {
23
24         private final int dotSize;
25
26         public AntPanel(int dotSize) {
27             int pixels = dotSize * world.size;
28             this.dotSize = dotSize;
29             setPreferredSize(new Dimension(pixels, pixels));
30             setBackground(Color.WHITE);
31         }
32
33         @Override
34         protected void paintComponent(Graphics g) {
35             super.paintComponent(g);
36             g.drawString("Steps:␣"+ant.steps, 10,10);
37             g.drawString("Is␣the␣ant␣building␣a␣highway?", 10, 25);
38             // we draw the cells
39             for (int x = 0; x < world.size; x++) {
40                 for (int y = 0; y < world.size; y++) {
41                     if (world.getCellColor(x, y) == World.Color.BLACK) {
42                         g.setColor(Color.BLACK);
43                         g.fillRect(dotSize * x, dotSize * y, dotSize, dotSize);
44                     }
45                     if (world.checkHighwaySouth(x, y)) {
46                         g.setColor(Color.GREEN);
47                         g.drawString("Yes,in␣direction␣South-East", 190, 25);
```

```
48                  }
49                  else if (world.checkHighwayNorth(x, y)) {
50                      g.setColor(Color.GREEN);
51                      g.drawString("Yes,in␣direction␣North-West", 190, 25);
52                  }
53                  else if (world.checkHighwayEast(x, y)) {
54                      g.setColor(Color.GREEN);
55                      g.drawString("Yes,in␣direction␣North-East",190, 45);
56                  }
57              }
58          }
59      }
60  }
61
62      public LangtonAnt() {
63          // we create the world with 200 size for w and h
64          world = new World(200);
65          world.setCellColor(100, 100, World.Color.BLACK);
66
67          // we set the Ant at the good position
68          ant = new Ant(100,100, Ant.Direction.SOUTH);
69          System.out.print(ant.steps);
70          frame = new JFrame("Langton's␣Ant");
71          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
72          frame.setLayout(new FlowLayout());
73          antPanel = new AntPanel(3);
74          frame.add(antPanel);
75          frame.pack();
76          frame.setVisible(true);
77      }
78
79      // method for starting the world evolution
80      public void run(int maxSteps, long delay, long period) {
81          Timer timer = new Timer();
82          timer.schedule(new TimerTask() {
83
84              @Override
85              public void run() {
86                  // we stop the evolution when max steps for the ant is reached
87                  if ((ant.steps > maxSteps ) && ant.isInWorld(world)) {
88                      timer.cancel();
89                  } else {
90                      // we repaint the panel
91                      SwingUtilities.invokeLater(() -> {
92                          antPanel.repaint();
93                      });
94                      // the ant makes a step in the world according to the rules of the
                            Langton's Ant
95                      ant.step(world);
96
97                  }
98              }
99          }, delay, period);
100     }
101
102     public static void main(String[] args) {
103         // Now, we have to display the UI and launching the world evolution
104         SwingUtilities.invokeLater(() -> {
```

```
105             LangtonAnt ant1=new LangtonAnt();
106             ant1.run(15000, 400, 5);
107         });
108     }
109 }
```

We changed some part of the code. We created the functions to check if the ant builds a **highway**, named **checkHighwaySouth()**, **checkHighwayNorth()** and **checkHighwayEast()** in the class World.
We added line 37 and lines 45 to 56 in the class AntPanel to show the text, whatever the ant is building an highway or not.
We also added the code to display the steps of the ant. To show the steps count, we added the line 36 in the class AntPanel.
Later on, we changed the part of code, where the program stops if the ant touches the edge of the frame. We change it so, that the program will continue till all ant are out of our frame.
To realise it, we remove the second condition of the if-loop at line 87 in the class LangtonAnt. And we added at line 95, an if-loop to check whatever the ant is in our frame or not. If the ant is in our frame,then it goes into the if-loop.
And the code that we added at line 95:

```
1 if(ant.isInWorld(world)){
2     ant.step(world);
3 }
```

In next section we showed some result of our experiments.

4 Experiment and Performenence

4.1 Basical output

1 ant facing North,after 11 000 steps

4.2 Try with 2 ants

Firstly, we made for same starting point, but different direction:

2 ants facing South,North, same starting point, 5289 steps

2 ants facing South,South, same starting point,7836 steps

"

2 ants facing East,West

2 ants facing North,East

same starting point, 2 black squares, clears it and repeats it

4.2.1 20-pixel different for x coordinate

Now we have 2 ants where they are 20 pixel apart in x coordinate and same y coordinate

2 ants facing South,South, 6453 steps

2 ants facing South,North, 6788 steps

2 ants facing North, East, repeats itself

For *North, East* case, the pattern repeats itself. What we mean by repeating itself is that it clears itself and repeats the same pattern again even after 39013 steps.

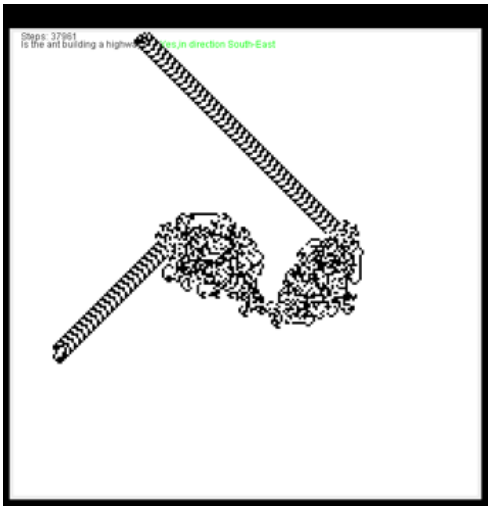
4.2.2 50-pixel different for x coordinate

Now we have 2 ants where they are 50 pixel apart in x coordinate and same y coordinate

10000 steps, 2 highways

13900 steps, starts to clear it

23720 steps, back to initial position



38000 steps, final result

2 ants facing West,North

Something very interesting happened : two ants firstly build the highway after the step of 10000, and then they turned back to the origin at the step of 23000, and start building a highway again at an another direction.

2 ants facing North, North, 13300 steps

2 ants facing South,North, 12700 steps

For north, north and south, north case, just one highway has been built, which is caused by our code, since the program stops when an ant touches the edge of the grid,(We made a little change on the code later on).

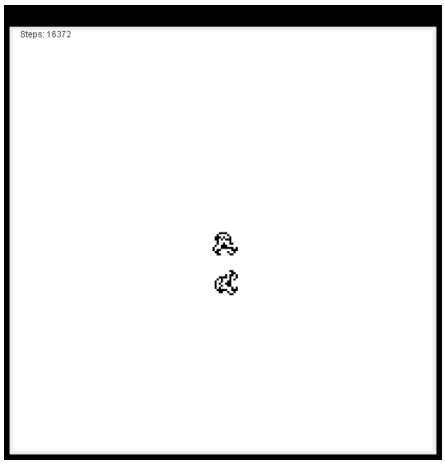
4.2.3 20-pixel different for y coordinate

Now we have 2 ants where they are 20 pixel apart in y coordinate and same x coordinate

2 ants facing North,South, 15000 steps

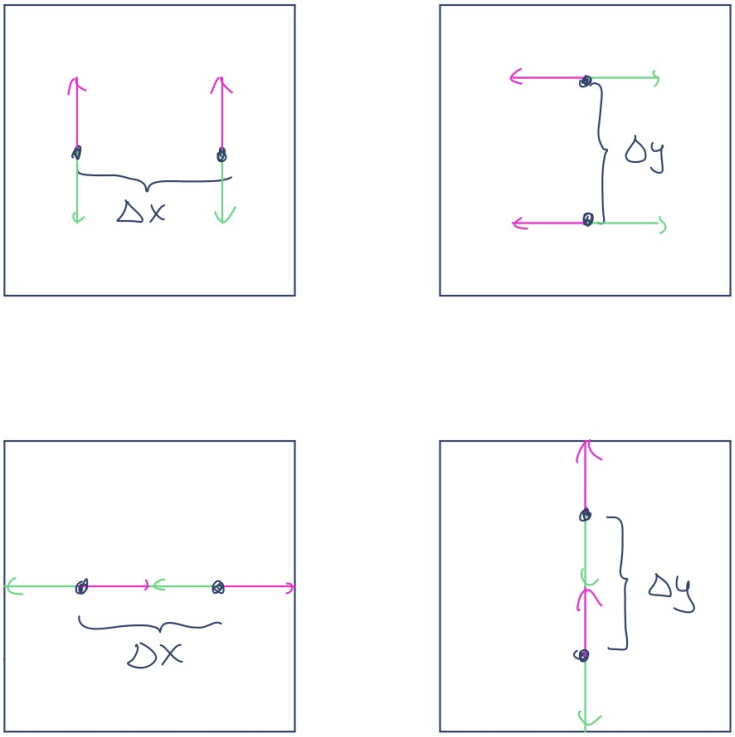
2 ants facing North,North, 11500 steps

For *north, south* and *north, north* case, we have noticed that it just builds one highway, this is caused by our code. When an ant touches the edge of the grid, the program stops automatically.(We made a little change on the code later on).



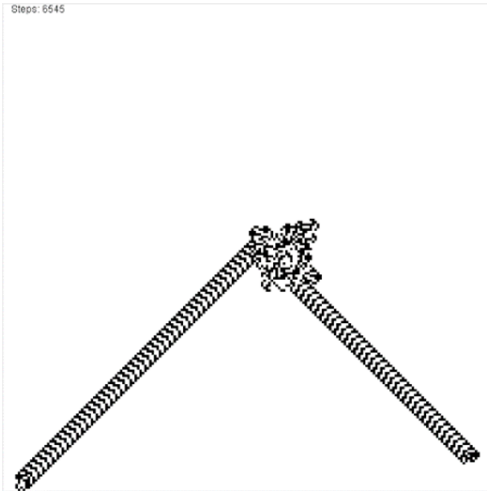
2 ants facing North,West, 16500 steps

build a chaotic pattern, clear it and repeat it again

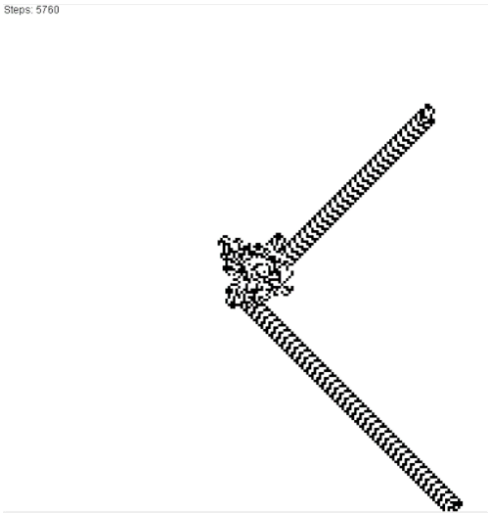


Having two ants facing to *north, north* or *south, south*, that are a-pixel apart in x-coordinate and two ants facing to *east, east* or *west, west*, that are a-pixel apart in y-coordinate should have the same result but rotated. And also for two ants *east, east* or *west, west*, that are a-pixel apart in x-coordinate and two ants facing to *north, north* or *south, south*, that are a-pixel apart in y-coordinate should also be the same but just rotated.

Here is an example:

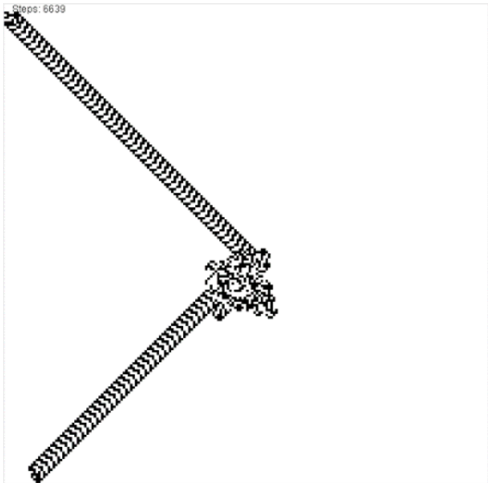


2 ants facing south, south, $\Delta x = 20$



2 ants facing east, east, $\Delta y = 20$

2 ants facing north, north, $\Delta x = 20$



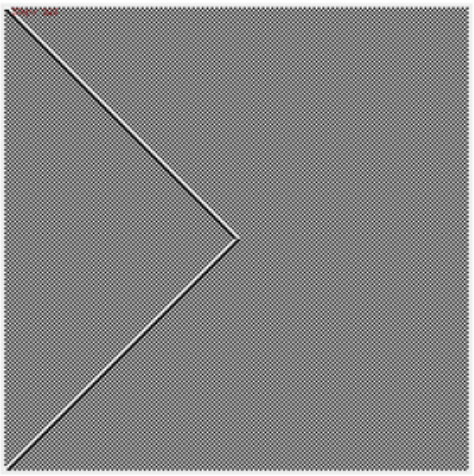
2 ants facing west, west, $\Delta y = 20$

As we can see, the highways are just rotated.

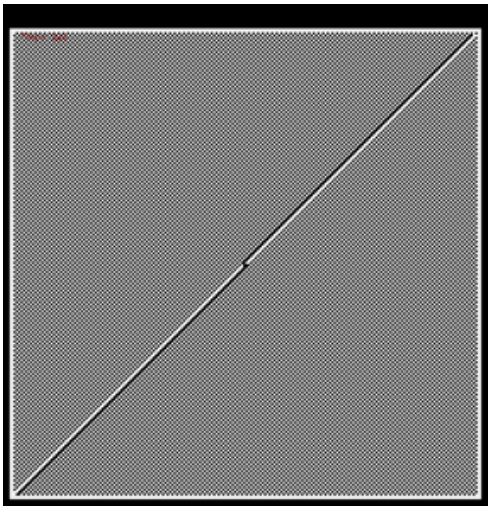
4.2.4 Checkerboard

Some interesting idea, if we keep two ants, but we do some changes to the grid.
For Example, *Checkerboard*:
To initialise the grid to a checkerboard, we added some lines of code in the constructor World of the class World.
And here is the code:

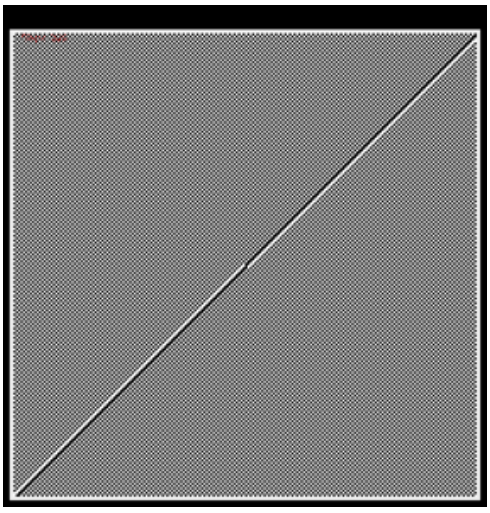
```
1    for (int x = 0; x < size; x++) {
2        for (int y = 0; y < size; y++) {
3            if((x+y)%2==0){
4                cells[x][y] = Color.WHITE;
5            }
6            else {
7                cells[x][y] = Color.BLACK;
8            }
9        }
10   }
```



2 ants facing North, West



2 ants facing North,South



2 ants facing North,North

4.3 Multiple ants

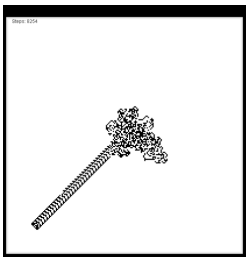
Now, we would concentrate on finding results for multiple ants, where they have the same starting point.
Then, we are going to enumerate all the case for the different amount of ants.

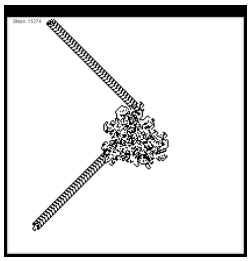
4.3.1 3 ants

3 ants facing North, North, North

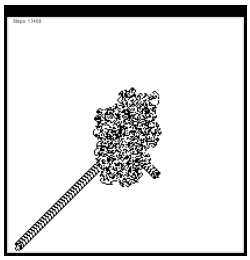
3 ants facing North, East, West

3 ants facing North, East, East



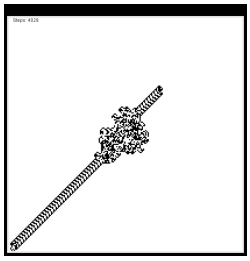


3 ants facing North, West, West

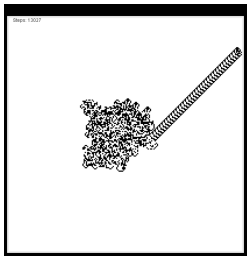


3 ants facing North, South, South

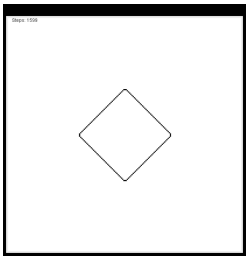
4.3.2 4 ants



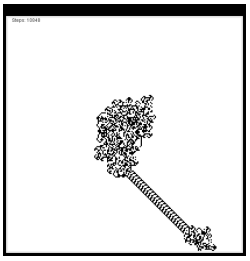
4 ants facing South, South, South, South



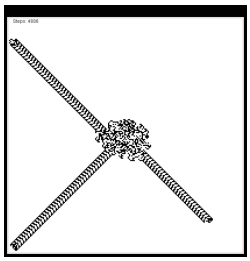
4 ants facing south, south, south, west



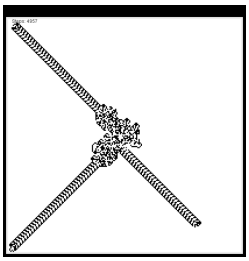
4 ants facing south, south, west, west



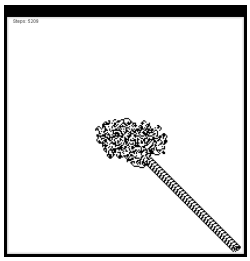
4 ants facing south, west, west, west



4 ants facing south, south, south, north



4 ants facing south, south, north, north



4 ants facing south, north, north, north

4 ants facing west, north, north, north

4 ants facing west, north, west, north

4 ants facing west, north, west, west

4 ants facing west, north, west, east

4 ants facing west, east, west, east

4 ants facing east, east, west, east

4 ants facing east, east, east, east

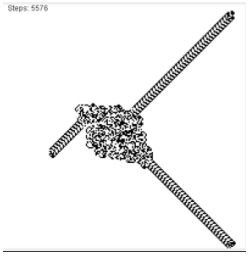
4.3.3 5 ants

5 ants facing east, east, east, east ,east

5 ants facing west, east, east, east ,east

5 ants facing west, west, east, east ,east

5 ants facing west, west, west, east ,east

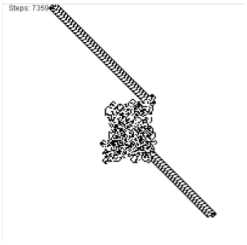


5 ants facing west, west, west, west ,east

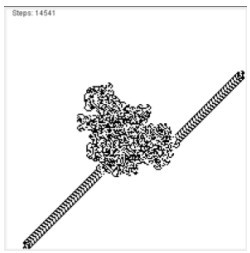
5 ants facing west, west, west, west, north

5 ants facing west, west, west, north, north

5 ants facing west, west, north, north, north



5 ants facing west, north, north, north, north



5 ants facing west, north, north, north,east

5 ants facing west, east, north, north ,east

5 ants facing west, east, north, east, east

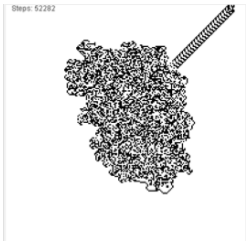
5 ants facing north, east, north, east, east

4.3.4 6 ants

6 ants facing north, north, north, north, north, north

6 ants facing north, north, north, north, north, west

6 ants facing north, north, north, north, west, west



6 ants facing north, north, north, west, west, west

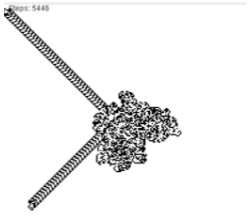
6 ants facing north, north, west, west, west, west

6 ants facing north, west, west, west, west, west

6 ants facing north, south, west, west, west, west

6 ants facing north, north, north, north, north, south

6 ants facing north, north, north, north, south, south



6 ants facing north, north, north, south, south, south

6 ants facing south, south, south, east, east, east

4.3.5 8 ants

8 ants facing-south, south, south, south, east, east, east, east

8 ants facing south, south, south, south, west, west, west, west

8 ants facing north, north, north, north, west, west, west, west

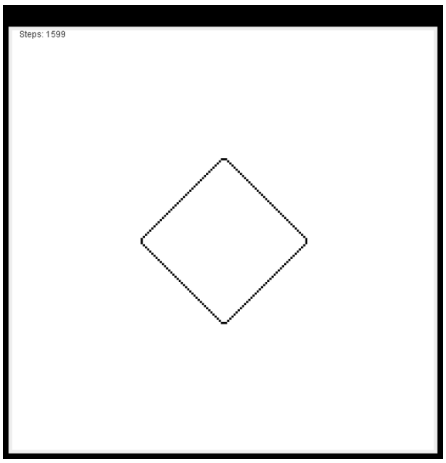
We stop at 8 ants, it's probably enough for the observation.

During the enumeration above, we found for some conditions, we receive the same results. For example, the rectangle form. Next one, we would like to see in which case we will have the rectangle form result.

4.4 Rectangle Form

By referring the enumeration above, we see the rectangle form if we have 4 ants. So here are the possibilities that we found:

- south, south, east, east
- south, south, west, west
- north, north, east, east
- north, north, west, west



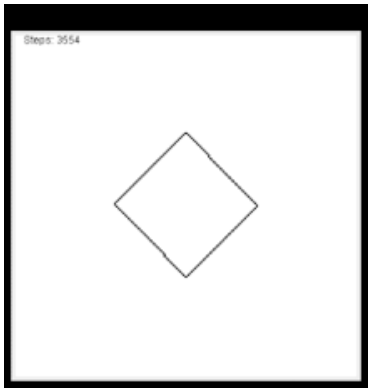
4 ants, same starting point

Then we think if the amount of the ants are even, we probably get the same results for $6, 8, 10 \ldots$. But unfortunately it's not the case for them, we didn't find any output as expected if we set the amount of ants as $6, 8, 10 \ldots$.

4.5 Differential position

Now we fixed the amount of the ants as 2 and the initial directions of the ants as *south, south*, but only change the difference of position between both ants.

Here we found another rectangle form but the initial directions are *south, south* and the two ants are 1 pixel apart in x-coordinate.



$\Delta x = 1$, growing rectangle

$\Delta y = 1$, black squares turns back to white, then repeats

$\Delta x = 2$

$\Delta y = 2$

$\Delta x = 3$, infinite loop, black sqaures will be erased,then it repeats

$\Delta y = 3$, infinite loop

$\Delta x = 4$

$\Delta y = 4$

$\Delta x = 5$, infinite loop

$\Delta y = 5$, infinite loop

$\Delta x = 6$

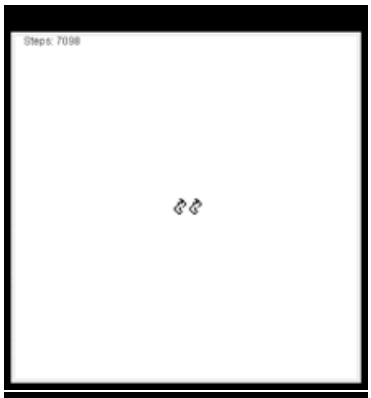
$\Delta y = 6$

$\Delta x = 7$, infinite loop

$\Delta y = 7$, infinite loop

$\Delta x = 8$

$\Delta y = 8$



$\Delta x = 9$, infinite loop

$\Delta y = 9$, infinite loop

$\Delta x = 10$

$\Delta y = 10$

$\Delta x = 11$, infinite loop

$\Delta y = 11$

$\Delta x = 12$

$\Delta y = 12$

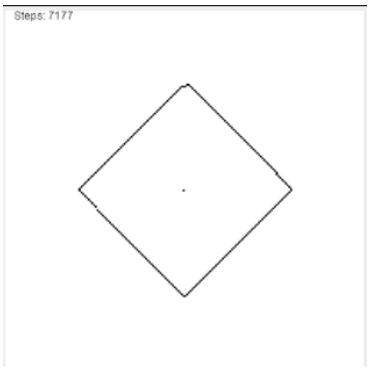
$\Delta x = 13$, infinite loop

$\Delta y = 13$, infinite loop

We realised that if the difference in x or y coordinate is odd, it does not build a highway Furthermore, it´s an infinite loop, since it builds a chaotic pattern, clears it and repeats it. Except when we have $\Delta y = 11$, in this case, the two ants form two highways towards two different direction.
Then we change the initial direction of the ant to *South, West*.

$\Delta x = 1$

$\Delta y = 1$



$\Delta x = 2$, growing rectangle

If we zoomed in, we can see that the upper corner is a little bit deformed and it has a

black point in the midlle of rectangle.

$\Delta x = 2$, upper corner, zoomed in

Same as before, we try it till $\Delta x = 13$ and $\Delta y = 13$.

$\Delta y = 2$,infinite loop, these black sqares will turned white again, then repeats it

We observed that now it is the opposite. When the difference in x or y coordinate is even, it does not build a highway. It will be in an infinite loop, but the pattern that the ants build are not the same.

Then we realized that according to our code, the program would stop when a highway touches the edge of the frame. And we wish to see what will happen to others ants. Therefore, we made a little change to the code, and here is what we found as output.

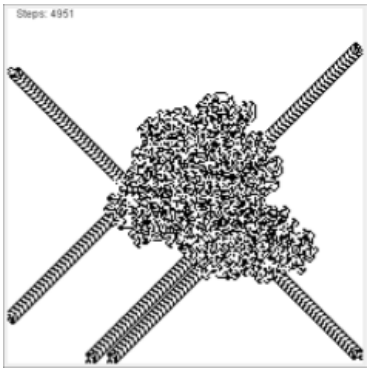
The idea is: If an ant is out of frame , it will stop at that position. The other ants will still move until they are also out of frame. These ants have all the same starting point and direction, which is *North*.

2 ants

3 ants

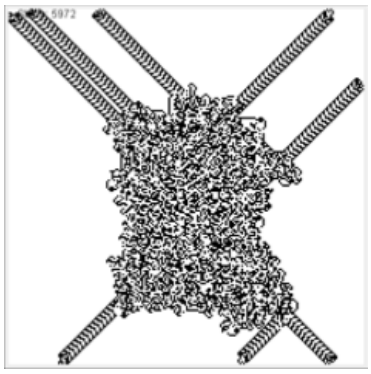
4 ants

5 ants



6 ants

7 ants

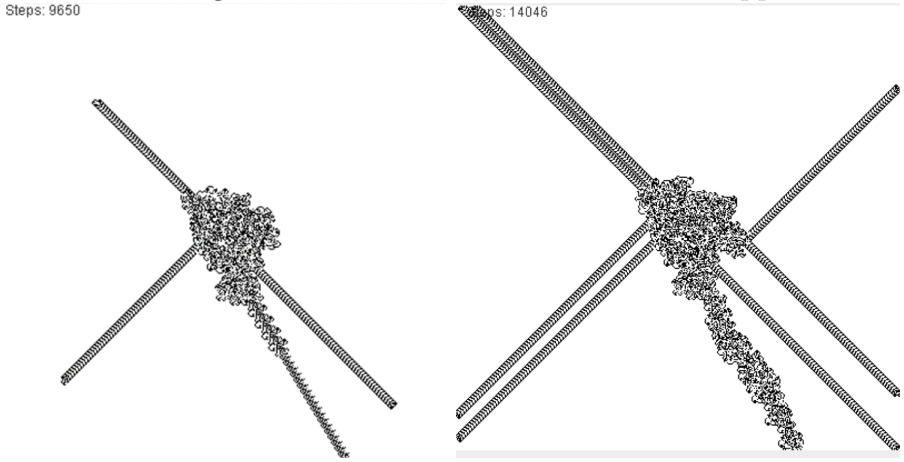


8 ants

9 ants

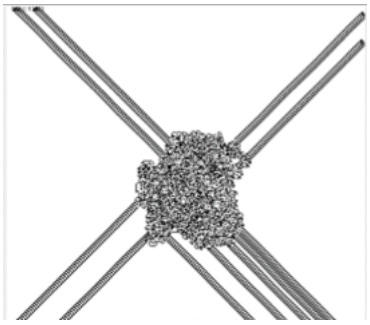
10 ants

And then we enlarge the frame size, and see what will happen:

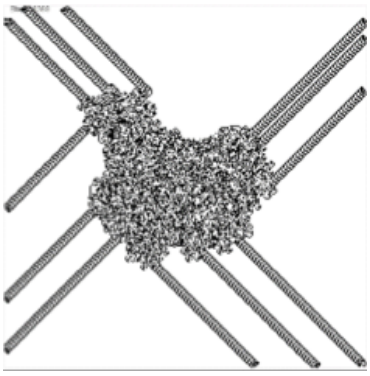


10 ants, 7 highway , 1 different type of highway

This different type of highway was formed by three ants. The first ant goes down to this direction and forms an another type of highway, that we can see in the first picture, on the second half of the "special" highway. Then a second goes also down to this direction, the "highway" can also be seen in the first picture on the first half of the "special" highway. Lastly a third ant builds the "special" highway that we see in the end, like shown in the second picture.

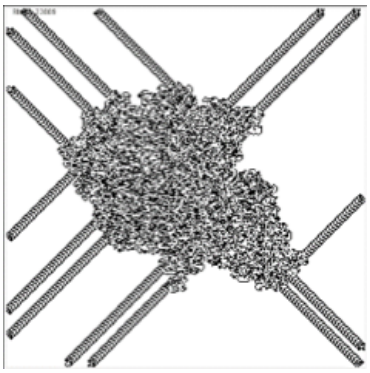


11 ants

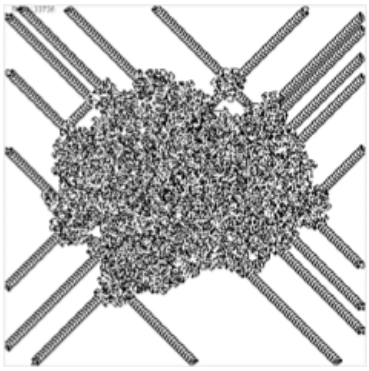


12 ants

13 ants



14 ants



15 ants

As we can see, the number of ants is directly proportinal to the number of highway,it has a ratio of 1:1. Which mean **number of ant = number of highway**,except when we have 10 ants. As we see on the picture above, there are only 7 highways and one special "highway", what we mean it is a different type of highway than usual.

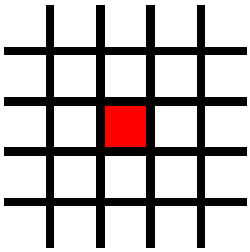
5 Some Extra

This section is under the environment of Matlab, just about some special thinking when we are doing the experiment. And unfortunately we are not able to find out the mystery beneath the result.

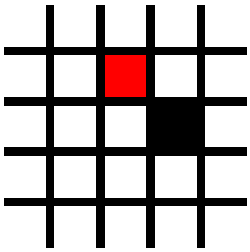
Since we know Langton's ant starts with chaotic but ends with ordered.

The thinking is about: if we let the ant moves as normal, but instead of changing the pixel under the ant, we change the pixel besides the ant. For example, always the pixel of the west or east. And we are curious about whether or not the ant would move chaotically and then build the highway.

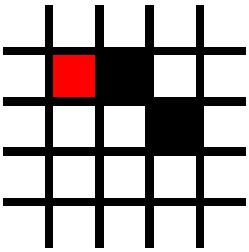
We start with changing the pixel at the *East* of ant, and initialise the direction as *North*.



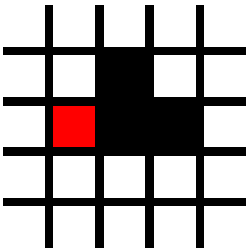
1st step of the ant



Then the ant goes to the north, and change the color of the pixel of the east

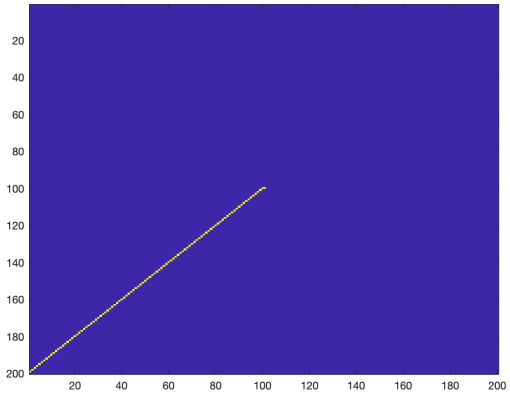


The pixel under the ant was white, therefore the ant turn left and move forward

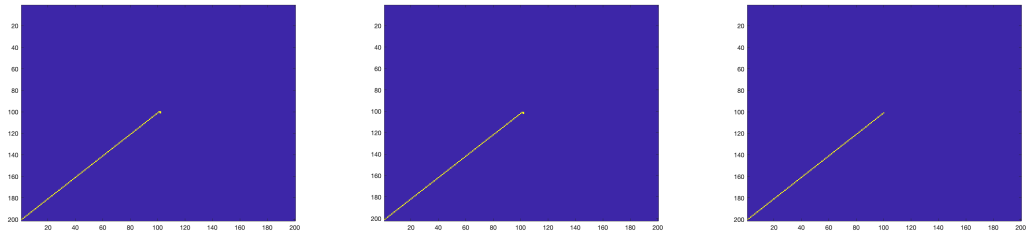


Again, the color was white, then the ant turn left

We then run it in matlab, and we have:

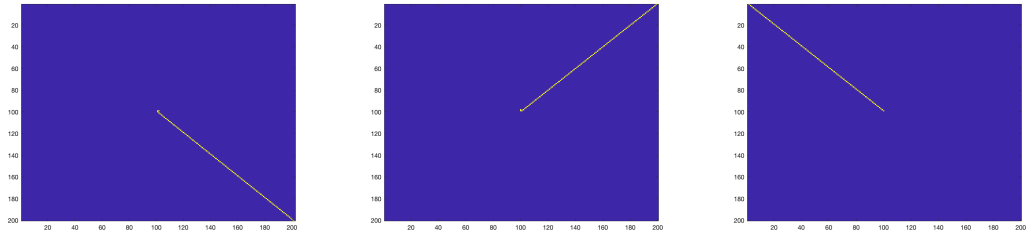


We repeat this idea with other three directions:



We noticed that the ant lose its randomness. And it moves immediately to the same direction, *except, depending on the initial direction of the ant, the ant would firstly turn around then moves directly.* Now we fix the direction, and only change the pixel which is affected by the ant.

Since we have done the pixel of the *East*, we now do the same thing for the pixel of the *North, West, South*.



We see, it doesn't affect a lot, the only difference is the directions of the "line".

We've also tried some different place where the pixel changes, the results are the same as expected: the ant lost its randomness and it will no longer move chaotically and eventually build the highway.

Little conclusion: According to the rule of Langton's ant, we know the ant will always change the color behind when having a movement. Since the first 10,000 movements are chaotic, this cause the changes of the color are also chaotic. But if we fixed the pixel that affects by the ant every steps, i.e. The ant will always change the color besides. This will cause the ant lose its randomness and build a line towards one direction.

6 Conclusion

Trough all the experiment we did, we came to the conclusion that the ants do not always build a highway. Furthermore, there are three kind of "output", an highway, a growing rectangle or an infinite loop.
When we have two ants facing South, South having different staring point, we get two different "results" :

- build an highway when the x or y coordinate is even
- goes into an infinite loop, does not build an highway when the x or y coordinate is odd

And when we change the initial direction to South, West, we get the opposite, which means:

- build an highway when the x or y coordinate is odd
- goes into an infinite loop, does not build an highway when the x or y coordinate is even

Lastly, we noticed that if all the ants starts at the same position, the number of ants equals to the number of highway, except when we have 10 ants.

Later when we change the initial set of rules, instead of changing the color under the ant, we change the color besides the ant, and we get a different kind of "highway".
It´s always a line towards one direction of the following direction :

- Northwest
- Northeast
- Southeast
- Southwest

By setting the initial grid to a checkerboard, the ants also build a line towards the directions listed above.

7 References

References

[1] Christopher G. LANGTON *STUDYING ARTIFICIAL LIFE WITH CELLULAR AUTOMATA**, Department of Computerand CommunicationSciences, The Universityof Michigan, Ann At'hot; MI 48109, USA

[2] Sylvain Saurel
`https://ssaurel.medium.com/implementing-langtons-ant-in-java-with-a-swing-ui-e5fa57`