University of Luxembourg

Experimental mathematics 3

Turmites

Authors:
Oliver Jack
Francesca Feyereisen
Noah Thielen

Supervisor:
Ms. Tara Trauthwein

May 31, 2022

Abstract

The following paper is a detailed summary of our research about "Turmites", a special case of a so-called cellular automaton. Based on our findings, we will establish several conjectures. The main purpose of this paper is to broaden our and our readers' understanding of the repetitive structure that can be found in the chaotic turmite world.
We would like to thank Ms. Tara Trauthwein for all of her help and guidance throughout our project.

Contents

1 Introduction

Let us start off by giving a brief definition of what a turmite is, followed by a precise example to help understand the basic evolution of such a simulation.

A turmite, whose name originates from the insect "termite", can be regarded as a small ant living on an infinite 2-dimensional grid of squares. The turmite has two particular attributes, which constantly vary over time: a state and an orientation. Meanwhile, the squares of the grid also have a fluctuating attribute, namely a colour. While travelling across the grid from cell to cell, the turmite leaves a clear trail behind, colouring the visited squares with numerous shades. These colours, along with the ant's movement, are entirely defined by the following set of rules and matrices.

1.1 General rules

Suppose we are working with m distinct states $\{0, \ldots, m-1\}$, n distinct colours $\{0, \ldots, n-1\}$ and 4 possible turns/changes of direction $\{0 = \text{no turn}, 1 = 90^\circ \text{ turn to the right}, 2 = 180^\circ \text{ turn}, 3 = 90^\circ \text{ turn to the left}\}$ for our turmite. Next, let us define the 3 rule matrices, which will give us the new state and direction of our turmite, as well as the new colour of the square the turmite leaves behind, after each single step:

$$S = \overset{n \text{ columns} \longrightarrow}{\begin{pmatrix} s_{0,0} & s_{0,1} & \cdots \\ s_{1,0} & s_{1,1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} \begin{matrix} m \\ \text{rows} \\ \downarrow \end{matrix} \quad \text{with } s_{i,j} \in \{0, ..., m-1\}, \forall i \in \{0, ...m-1\}, j \in \{0, ..., n-1\}$$

$$C = \overset{n \text{ columns} \longrightarrow}{\begin{pmatrix} c_{0,0} & c_{0,1} & \cdots \\ c_{1,0} & c_{1,1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} \begin{matrix} m \\ \text{rows} \\ \downarrow \end{matrix} \quad \text{with } c_{i,j} \in \{0, ..., n-1\}, \forall i \in \{0, ...m-1\}, j \in \{0, ..., n-1\}$$

$$T = \overset{n \text{ columns} \longrightarrow}{\begin{pmatrix} t_{0,0} & t_{0,1} & \cdots \\ t_{1,0} & t_{1,1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} \begin{matrix} m \\ \text{rows} \\ \downarrow \end{matrix} \quad \text{with } t_{i,j} \in \{0, 1, 2, 3\}, \forall i \in \{0, ...m-1\}, j \in \{0, ..., n-1\}$$

At each timestep, the turmite, of state s and direction d, finds itself on a square of colour c. Then, these three values are updated simultaneously in the following way:

$$s = S_{s,c} \qquad \text{(the } (s,c) \text{ coefficient in the state matrix S)}$$

$$c = C_{s,c} \qquad \text{(the } (s,c) \text{ coefficient in the colour matrix C)}$$

$$t = T_{s,c} \qquad \text{(the } (s,c) \text{ coefficient in the turn matrix T)}$$

Finally, the ant, of updated state s and direction d, moves one step in the direction that

it is facing, colouring in the cell it left behind with the new colour c. This procedure can then be repeated and will run on forever, unless one sets a boundary to the grid size or to the number of steps.

1.2 Basic example

Let us take a closer look at a specific example of a 2-state, 2-colour turmite to get a better understanding of how the grid evolves. The pictures shown down below were custom-made by us and can be seen as a visual aid to help understand the general rules.

Let

$$S = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$T = \begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix}$$

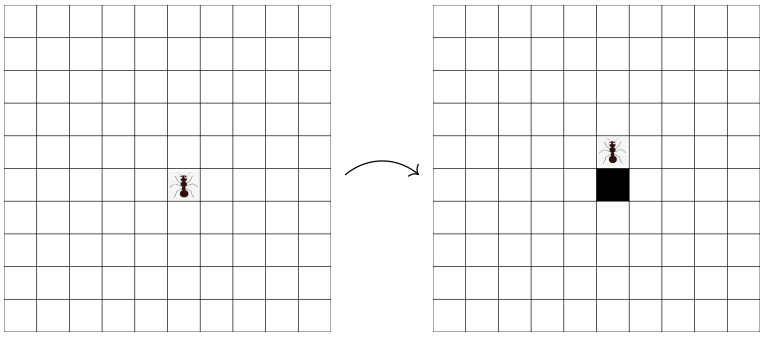
be the different rule matrices and let all the squares of the grid be of colour 0 (= white in our case) initially. If our ant is in state 0, facing North and sitting on a square of colour 0, then these three values are updated as follows:
After step 1:

$$s = S_{0,0} = 1$$

$$c = C_{0,0} = 1$$

$$t = T_{0,0} = 0$$

This means that the new state of our ant is 1, it's direction remains the same (since 0 represents no turn) and it moves one square to the North, colouring the square it leaves behind in the colour 1 (= black in our case). The new square the ant has moved to initially has colour 0.



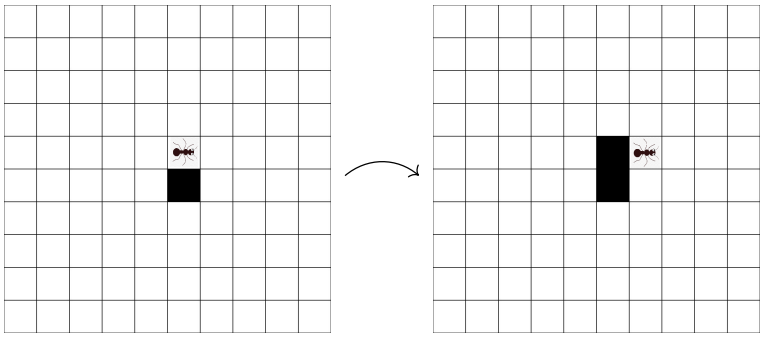
After step 2:

$$s = S_{1,0} = 1$$

$$c = C_{1,0} = 1$$

$$t = T_{1,0} = 1$$

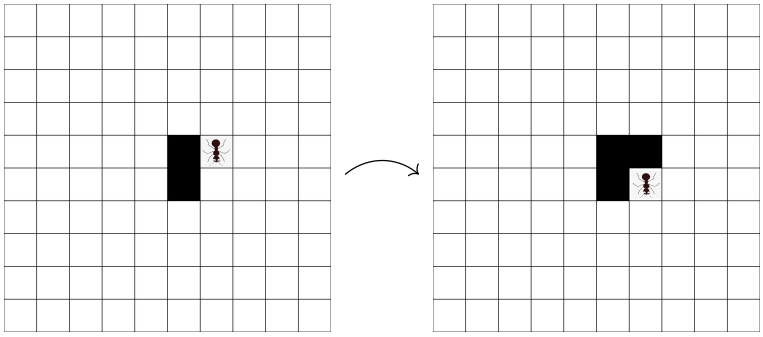
This means that the state of our ant remains 1, it turns 90° to the right and moves one square to the East, colouring the square it leaves behind in the colour 1.



After step 3:

$$s = S_{1,0} = 1$$
$$c = C_{1,0} = 1$$
$$t = T_{1,0} = 1$$

We get the exact same movement as described before.



By repeating this procedure, we obtain a nice, forever growing spiral, seen down below.

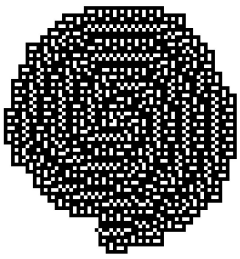


Figure 1: Visual representation after 10˙000 steps (image from own simulation)

To help understand where the idea of turmites originates from, we will first take a brief look at the history of different types of cellular automata and give a short summary of some already well-known mathematical results. Next, we will share our observations of the general behaviour of turmites, when they are defined by random rule matrices of various sizes. In addition, we will dive into further detail by explaining two phenomenons, which occur very frequently: the highway and the busy beaver. Furthermore, we will also analyse other interesting statistics such as the average steps it takes to notice a repetitive

pattern, or the average length of a continuous cycle. Moreover, we will examine every possible 2-state, 2-colour turmite, to try and find any particular properties about the rule matrices. Besides that, we will show the most remarkable shapes and patterns that we discovered during our research period. Last but not least, we will present our final conclusion and display the Python codes we used for our simulations in the appendix, followed by our references.

2 History & mathematical background

In computer science, turmites are known to be a type of cellular automaton and it has been proven that generally turmites are equivalently powerful as Turing machines, a mathematical and computational abstract machine, whose actions are defined by an internal state transition table.[8]

2.1 Langton's ant

In 1986, the computer scientist Christopher Langton became one of the first people to study the behaviour of turmites in general.[4] In particular, he studied a very popular case, known nowadays as "Langton's ant".[7] The movement of the ant can be defined as following:

- at a white square, turn 90° to the left, move forward one square and change the colour of the previous square to black
- at a black square, turn 90° to the right, move forward one square and change the colour of the previous square to white

This can be translated to these three rule matrices

$$S = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$T = \begin{pmatrix} 1 & 3 \\ 1 & 3 \end{pmatrix}$$

Here it is important to notice that there are multiple possible matrices that we could have used to define the same movement as described above. Hence, we can conclude that a turmite is not always defined in a unique way.

After about 10'000 steps, he was able to observe a repetitive pattern. This type of structure is frequently called a "highway" and is none other than a sequence of steps which repeats indefinitely. It turns out that this behaviour, where the turmite forms an infinitely long highway, is quite common among turmites, as will be explained in detail throughout the paper. It remains an unproven conjecture that, no matter the initial colour configuration of the squares of the grid, the ant will eventually build a highway in every case. Nevertheless, in 1997, the mathematician S. Troubetzkoy proved that, for all initial configurations of the grid, the trajectory of the ant is always unbounded.[5]

This property also holds if we would place the ant on a triangular grid. Furthermore, in 2000, three mathematicians, A. Gajardo, A. Moreira and E. Goles, were able to show that Langton's ant is Turing complete, meaning that it can be used to simulate any Turing machine.[2]

Figure 2: Visual representation of Langton's ant after about 8˙000, 10˙000 & 12˙000 steps (images from own simulation)

2.2 The Game of Life

Another famous cellular automaton is the so-called "*Game of Life*", invented by the mathematician John Conway in 1970. After having selected an initial configuration for the grid, the simulation requires no further input from its user and evolves over infinitely many steps, based on a set of well-defined rules, which were inspired by real life ideas. This sort of simulation was also shown to be Turing complete.[6]

Figure 3: Example of a configuration of the *Game of Life* (source: Bettilyon, T., `https://medium.com/tebs-lab/optimizing-conways-game-of-life-12f1b7f2f54c`)

3 Random rule matrices

Let us start off by analysing what happens when we define our turmite using randomly generated rule matrices. In order to keep a better overview of the different outcomes, we decided to primarily work with square matrices, i.e. turmites who have the same number of states as colours.

Naturally, there were a few simple questions we asked ourselves at the beginning, regarding the general behaviour of our turmite:

- What types of shapes and growth patterns can our turmite possibly create?
- Will our turmite always end up forming a repeating pattern eventually, no matter which rule matrices are used to define it?
- Does the frequency with which each pattern occurs depend on the size of the rule matrices?
- In general, how many steps does it take to reach a repetitive cycle and how many cells will our turmite have visited?
- On average, how often does our turmite visit each individual square and how long are the continuous cycles?

To fully understand our experimental results, we first need to lay the groundworks, by giving some necessary definitions, which we will frequently use to describe the three possible movement outcomes a turmite can have. It is important to notice that these definitions are by no means the "unique correct" definitions of these terms, rather they allow us to describe certain ideas in a simple and precise way.

3.1 Definitions

Definition 3.1.1 (Highway)**.** A highway can be defined as a periodic cycle of n ($\in \mathbb{N}$) steps, for which there exists $N \in \mathbb{N}$ st. for all steps after the N^{th} step, the following conditions hold true:

- $\forall k \geq N$, the state and direction of the turmite after k steps and after $k + n$ steps is the same
- $\forall k \geq N$, the colours of the squares the turmite is sitting on after k steps and after $k + n$ steps are equal

Definition 3.1.2 (Busy beaver)**.** A busy beaver (first studied by computer scientist Allen H. Brady [1]) is also a periodic cycle of n ($\in \mathbb{N}$) steps, which shares the exact same properties as a highway, however, with one additional condition needing to be verified:
$\exists N \in \mathbb{N}$ st. $\forall k \geq N$, the square on which the turmite is located after k steps and after $k+$ n steps is identical

An interesting consequence of this property is that after the N^{th} step, the turmite no longer visits any new, unvisited squares. In some cases it even appears as if the turmite is "frozen" or completely static.

Definition 3.1.3 (Random movement)**.** We classify the movement of our turmite as random, if neither a highway, nor a busy beaver is formed after a certain amount of steps.

3.2 Visual simulations vs analytical simulations

To be able to acquire sufficient conclusive data, we made use of two particular types of simulations: visual simulations and analytical simulations. (The two principal Python codes that were used, as well as further data can be found in the appendix of our paper).

On the one hand, the visual simulations enabled us to make some preliminary assessments, as to how the movement of our turmite would evolve over time. With the help of the Python extension *Pygame*, we were able to project the simulation on to the screen and take screenshots over regular step intervals.

On the other hand, the analytical simulations gave us a much better understanding of how the turmite would behave on average. Since our program allowed us to run through a large number of cases (default value: 10˙000 times) of different rule matrices, we were able to gather useful statistics, which we then compared with our initial assumptions. All of the statistics that we obtained during these simulations were saved in EXCEL files, in order to refer to any data if necessary.

3.3 Visual examples

Next, we will show some visual examples of the simulations we ran through. Here we worked with a 10-state, 10-colour turmite on 100×100 grid and we specifically chose the following three examples to illustrate the different possible outcomes.

Figure 4: Turmite forming a highway after 1˙000, 2˙000 & 3˙000 steps respectively (images from own simulation)

Here we can see that our turmite seems to move chaotically over the first 1˙000 steps, before its movement becomes regular and forms an infinite highway, which eventually exits the grid.

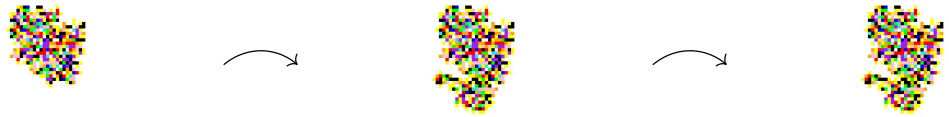


Figure 5: Turmite forming a busy beaver after 2˙000, 6˙000 & 10˙000 steps respectively (images from own simulation)

Once more, we can see that our turmite tends to grow randomly in all directions at the beginning, until it becomes "stuck" in an infinite loop, in which it stays for the rest of the simulation. This explains why the second and third image are identical.

Figure 6: Turmite moving randomly after 2˙000, 6˙000 & 10˙000 steps respectively (images from own simulation)

In this case, we can see that our turmite builds neither a highway, nor a busy beaver after 10˙000 steps. However, this does not exclude the possibility of it ending up in one of these two types of cycles. In fact, it is an open question in mathematics, as to whether or not all turmites form a highway or busy beaver eventually. This is almost impossible to check, since we are unable to run every possible case infinitely long. Nevertheless, we will try to tackle this problem for the 2-state, 2-colour turmite later on in the paper.

3.4 First observations

After having simulated over 1˙000 turmites, running for 10˙000 steps and defined by random rule matrices ranging in sizes from 2×2 to 10×10, we came up with our first observations based on the visual results:

- Throughout our simulations, we were able to find multiple examples of each type of movement defined in 3.1, no matter the size of the rule matrices.
- The number of times the turmite formed a busy beaver tended to decrease as the matrix sizes grew, while the number of highways increased with respect to the

growing sizes. In general, most turmites ended up moving in cycles during the first 10˙000 steps, meaning that the amount of turmites whose movement can be described as random, was relatively low. Hence, one could guess that every turmite will eventually form either a highway or a busy beaver.

- On average, it seemed to take a larger amount of steps to reach a cyclic movement as the sizes of the matrices became bigger.
- For those turmites that eventually formed a highway, the associated cycles appeared to grow in size, i.e. the movement became periodic after larger number of steps, as the matrix size rised. In the cases of a busy beaver, this was usually harder to identify, since the movement of the turmite became quite static at a certain timepoint.

3.5 Frequency analysis of different outcomes

Following our first assumptions that were based on the collected visual data, we wanted to take our analysis a step further. By simulating 10˙000 random turmites on blank grids for all matrix sizes ranging from 2×2 to 10×10 and checking their outcome automatically, we were able to gather enough data to get a better understanding of the average turmite behaviour and the frequency of each possible outcome.

Let us start off by taking a look at the table which summarises the frequency of each of the three possible outcomes for different matrix sizes, after having simulated each single turmite for a maximum of 10˙000 steps. Here, one could also add subcategories to each of the three outcomes, however, we decided to focus primarily on these three main types of movement.

Table 1: Outcome of 10˙000 turmites after 10˙000 steps

States/Colours	Random movement	Highway	Busy beaver
2	1399	3280	5321
3	2169	3729	4102
4	2425	4172	3403
5	2578	4492	2930
6	2633	4829	2538
7	2781	4953	2266
8	2987	5112	1901
9	3093	5262	1645
10	3227	5384	1389
11	3333	5375	1292
12	3447	5455	1098

When comparing the analytical results with the visual ones, we can see that they overlap. Overall, we were able to observe that, on average, the proportion of turmites that move randomly or end up forming a highway tends to increase with the matrix sizes, whereas the busy beavers become less frequent. By plotting the outcome frequencies as a function of the matrix sizes, we obtain the following graph.

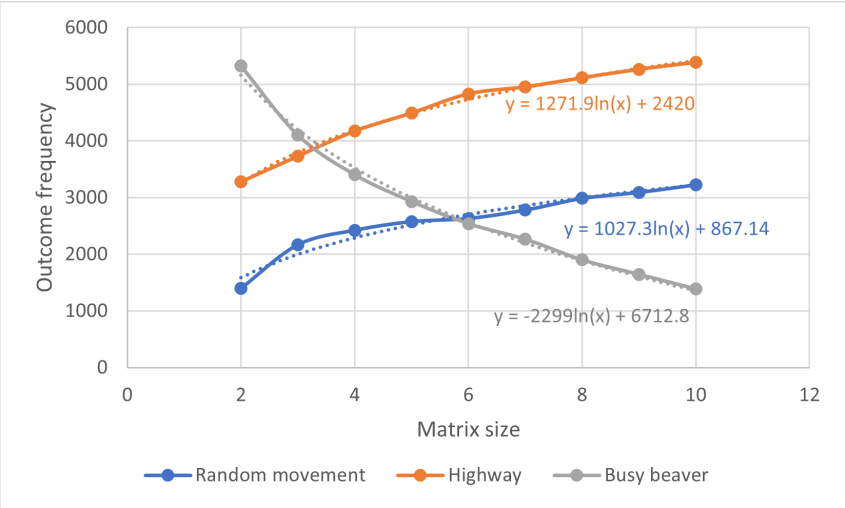


Figure 7: Outcome frequency with regard to matrix size (after 10˙000 steps)

Here, we notice that the functions related to the random movements and the highways both seem to have logarithmic growth. On the other hand, the graph associated to the busy beavers resembles the graph of the negative logarithmic function.

Next, we decided to increase the maximum number of steps to 100˙000, meaning that the movement of each individual turmite would be analysed for 10 times as many steps. We did this in order to see whether or not the total simulation steps plays a significant role or not. As a result, we could observe some clear differences compared to the previous case, especially with respect to the number of turmites whose movement can be classified as random.

Table 2: Outcome of 10˙000 turmites after 100˙000 steps

States/Colours	Random movement	Highway	Busy beaver
2	11	3968	6021
3	18	5005	4977
4	23	5549	4428
5	28	6051	3921
6	33	6336	3631
7	41	6865	3094
8	53	7096	2851
9	69	7566	2365
10	94	7643	2263

The main aspect that stands out in this table is the extremely low proportion of turmites that still moved randomly after 100˙000 steps. Given enough timesteps, the turmite will more often than not tend to form some sort of pattern or cycle eventually. This strengthens the assessment we made earlier, when we visually observed that most turmites, no matter the size of the rule matrices, could be ordered into two main groups: the highways and the busy beavers.

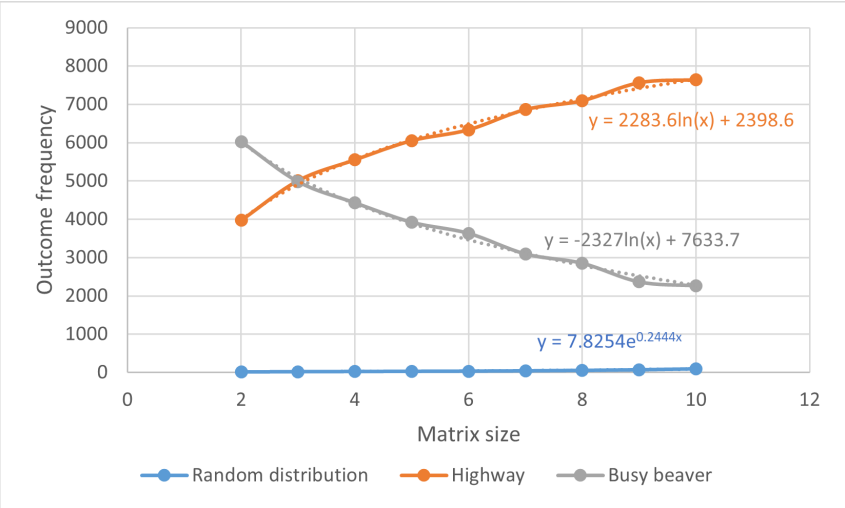


Figure 8: Outcome frequency with regard to matrix size (after 100˙000 steps)

Furthermore, we can perceive the growth of these two categories to be almost as equivalent as before. Meanwhile, after curve fitting the plotted data for the random movements, we found the exponential function to be the optimal approximation for our data. However, this is hard to identify with the graph above, since the exponent of the exponential function is relatively small compared to the growth scale of the other functions. With the help of this exponential function associated to the random movement of our turmite, we can already predict that the average number of steps it takes to reach a pattern correlated with the matrix size will rise. On the other hand, the growth of the two other functions simply tells us that the probability of a cycle being a closed loop (i.e. having the same start and end point) decreases as the matrix size increases.

In addition, we wanted to know if the initial colour configuration of our grid influences the proportions of the different outcomes in any way. Therefore, we computed as many random turmites as before, however, this time we distributed random colours to all the squares at the beginning. A first thing we realised was that these simulations took a lot longer to finish than those in the previous case. This is most likely due to the fact that the turmite "struggles" to form a repetitive cycle, since unvisited squares play a crucial role in determining the turmite's next state, colour and move. In contrast, the white squares in the previous case tended to act more "neutral", since the entire grid was initially filled with a unanimous colour.

The following table presents the results we obtained from our simulations:

Table 3: Outcome of 10˙000 turmites after 100˙000 steps over a randomly coloured grid

Matrix size	Random movements	Highways	Busy beavers
2	1865	0	8135
3	3300	0	6700
4	2444	0	7556
5	3748	0	6252
6	3622	0	6378
7	4327	0	5673
8	6333	0	3667
9	6425	0	3575
10	7704	0	2296

Unlike the previous case, not a single highway was detected. One explanation for this is that when a turmite tries to form a highway in a randomly coloured grid, it is constantly broken down because it repeatedly visits new squares of possibly different colours. In that sense, a grid of unanimously coloured cells clearly favours the formation of highways. Meanwhile, once a turmite is locked in a busy beaver, there is no possible way for it to escape afterwards. Therefore, these appeared with a similar frequency as before. Furthermore, the proportion of turmites which moved randomly tends to increase as the size of the matrices grow. However, this may be closely related to the total number of simulated steps. All in all, these differences simply show us that the unique behaviour and special properties of the turmite over a blank grid is by no means true over grids whose initial colour configuration is random.

3.6 Statistical analysis of other interesting properties

After having found out that the majority of turmites defined by random rule matrices end up forming a highway or a busy beaver, we wanted to see if we could determine any other interesting properties concerning the average behaviour of our turmite. The following analytical results are based on the simulation of 10˙000 random turmites on a blank grid of size 1˙000×1˙000 for each matrix size ranging from 2×2 to 10×10.

More precisely, we wanted to analyse the following characteristics:

- average number of steps it takes a turmite to reach a repetitive cycle
- average number of squares that are visited
- average number of times a visited square is revisited
- average length of a cycle

The table down below summarises the average conduct of our turmite in terms of these properties:

Matrix size	Avg steps until repetitive pattern	Avg squares visited	Avg times each square is visited	Avg cycle length
2 x 2	239.212	147.778	29.161	732.775
3 x 3	325.278	265.38	23.248	1284.224
4 x 4	554.57	467.621	14.903	2769.541
5 x 5	628.242	718.111	8.976	4538.06
6 x 6	3152.863	1126.408	6.559	5953.08
7 x 7	4339.802	1302.979	4.591	6287.292
8 x 8	5657.325	1565.61	3.844	7137.962
9 x 9	9930.249	2074.644	3.375	7291.55
10 x 10	13343.278	3774.762	2.807	7584.659

First of all, we notice that the average number of steps it takes until a pattern occurs with respect to the matrix size tends to grow exponentially. Moreover, this strengthens our previous prediction that this number increases as the sizes get bigger. Here, we use the convention that if no cycle is detected after 100˙000 steps, we simply say that it took this amount of steps to reach a repeating cycle.

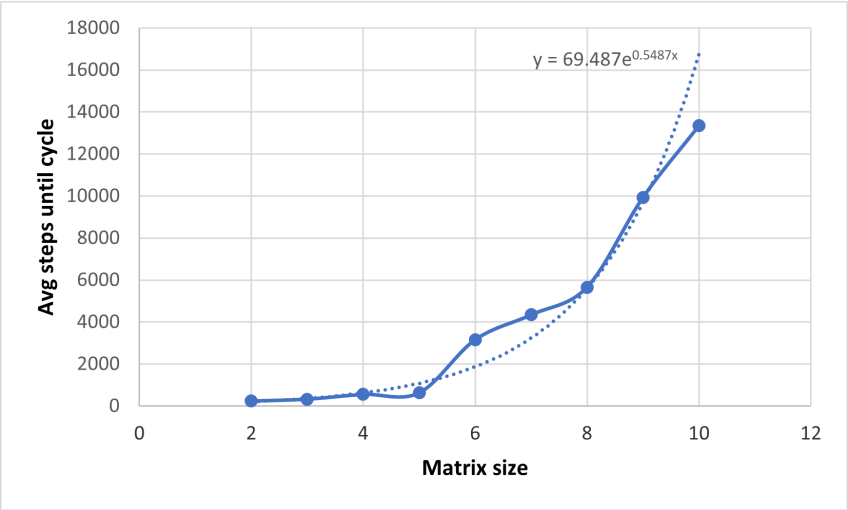


Figure 9: Average number of steps until cycle

Next, we can see that the average number of visited squares as a function of the matrix size behaves like the exponential function. This shows that the turmite's movement tends to become more expansive as the sizes of the rule matrices grow. Here it is important to remember that these results are related to a bounded grid, since it would be impossible to gather such conclusive data over an infinite grid. Since we found turmites of every matrix size that formed highways and since the proportion of highways tends to grow with respect to the sizes, we can conclude that, over an infinite grid, the average number of visited squares tends to infinity, no matter the matrix size.

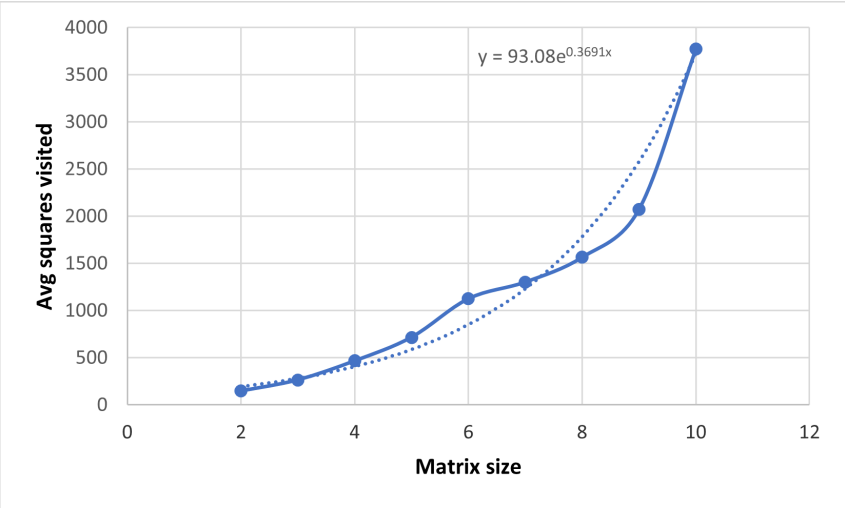


Figure 10: Average number of squares visited

In addition, we were able to observe that by plotting the average number of revisits to a visited square with regard to the matrix size, its graph resembles the graph of the inverse exponential function. This property is closely related to the previous one, as the turmite tends to cover more squares as the rule matrices get bigger. Since we kept the number of total simulated steps to be constant (100000 steps), this implies that revisits to visited squares become less frequent as the size increases.

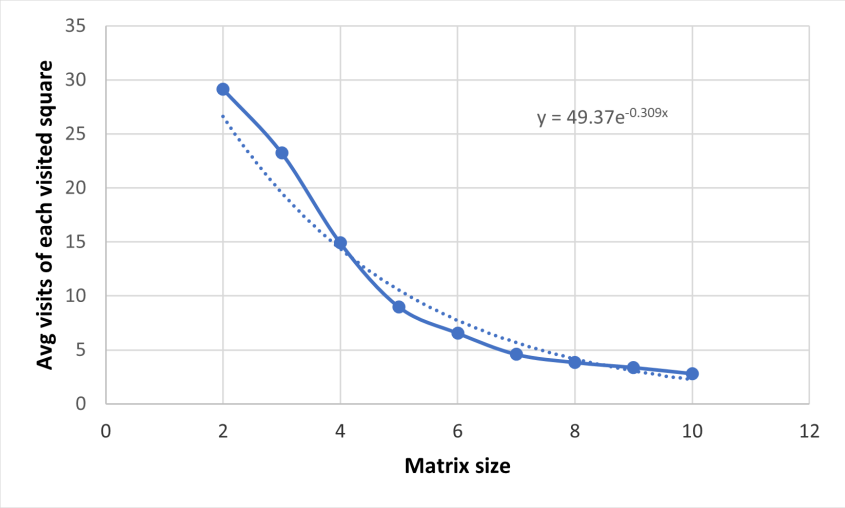


Figure 11: Average number of revisits to a visited square

Finally, by taking a look at the average cycle length as a function of the matrix size, we realise that it's growth is logarithmic. As we will see in the next section, the lengths of these cycles tend to have a large range of possible values. Here we used the convention that if no pattern is detected after 100˙000 steps, then we say that the cycle is of length 0.

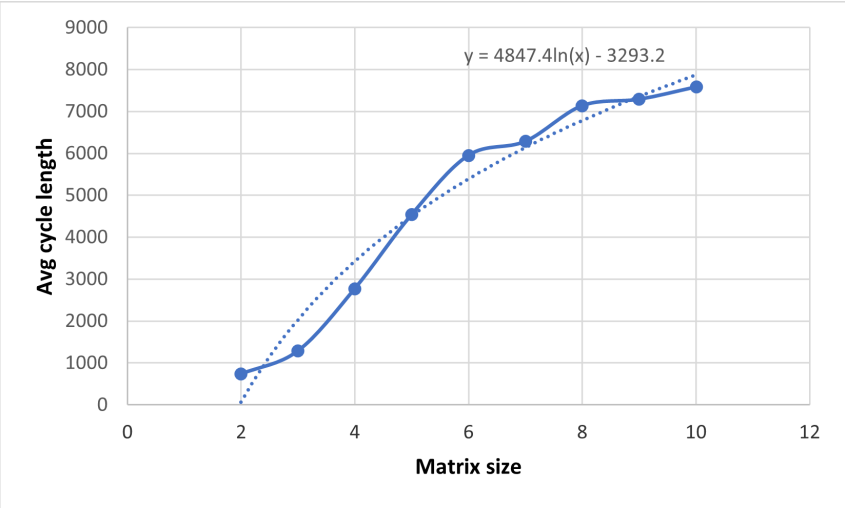


Figure 12: Average cycle length

3.7 Conjectures

Following the detailed analysis of our observations, we would like to come up with our very own conjectures, regarding the general behaviour of turmites defined by random rule matrices:

- Our main conjecture states that regardless of the matrix size, all turmites that are simulated over a blank grid will eventually form either a highway or a busy beaver, despite its chaotic movement at the beginning.
- The proportioning of the different possible outcomes largely depends on the total number of steps for which the turmite is simulated and on the initial colour configuration of the grid.
- On a blank grid, the proportion of highways grows logarithmically as the matrix sizes increase, while the proportion of busy beavers decreases slowly.
- On a grid with randomly coloured cells, the above properties don't hold in general. In this case, the turmite struggles to build any periodic cycles, especially highways.
- The average number of steps it takes to reach a cycle and the average number of squares visited grows exponentially with respect to the matrix size, i.e. the movement of the turmite becomes more expansive. However, this is only true if we are working on a bounded grid, otherwise the average number of visited squares will tend to infinity.
- The graph of the average revisits to a visited cell with regard to the matrix size resembles the inverse exponential function.
- On average, the cycle lengths tend to grow logarithmically as the rule matrices expand in size.

4 Further analysis on 2-state, 2-colour turmite

So far, we have only been working with turmites defined by random rule matrices of various sizes. We did this in order to deduce the average behaviour of the turmite. Now, we wish to shift our focus to the analysis of the 2-state, 2-colour case, by running through all possible turmites (total of $2^4 \cdot 2^4 \cdot 4^4 = 65536$) on a blank grid. Although it took quite a long time, we were able to simulate every single turmite visually, as well as analytically.

4.1 Simulation of every 2-state, 2-colour turmite

In a first phase, we analytically simulated all the possible turmites, on a blank grid of size $1'000 \times 1'000$, for a maximum of $10'000$ steps. Our program, which automatically checks for repetitive patterns, presented us with the following table:

Random movement	Highways	Busy beavers
80	28'438	37'018

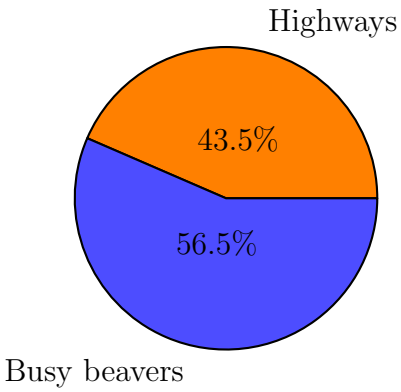
By comparing these numbers with those that we obtained for the randomly generated 2×2 turmites, we can see that the particular proportions are relatively similar. As before, busy beavers are more frequent than highways and we only get a low amount of turmites that do not create a cycle under these circumstances. Since the total number of turmites that moved randomly only represents a minor fraction of the total possible turmites ($\frac{80}{65'536} \approx 0.122\%$), we firmly believed that they would also form a highway or a busy beaver eventually. Therefore, we decided to compute these turmites again, this time on a larger grid ($10'000 \times 10'000$) and for more steps ($10'000'000$). After having simulated the 80 turmites under these new conditions, we got the following results:

Random movement	Highways	Busy beavers
0	76	4

Hereby, we believe to have shown a weaker version of our main conjecture for a very specific case:

Every single 2-state, 2-colour turmite on a blank grid will eventually begin to form either a highway or a busy beaver, which may go on for infinitely many steps

The exact proportions of the two cycles are the following:



Furthermore, this also shows that for every single one of these turmites, they reach a repeating cycle before 1˙000˙000 steps and before exiting a grid of size 10˙000 × 10˙000. Although these values are by no means the optimal or tightest bounds, they allow us to confine these two properties.

It is important to understand the significance of the statement "which **may** go on for infinitely many steps" in the conjecture that we stated above. With the help of our program, we have been able to check that at some point in time, every turmite will start to form a highway or a busy beaver. However, due to the nature of our automatic outcome checker, we cannot say with full certainty that these periodic cycles will repeat themselves infinitely often in an unobstructed direction, since it is impossible for our program to check for an infinite amount of steps. (Please refer to our analytical program in the appendix to get a better understanding of how this was implemented.) Therefore, we claim to only have shown a "weaker" version of our main conjecture.

4.2 Other interesting properties

In this paragraph, we will analyse what type of impact the invertibility, the symmetry and the diagonalisability of the rule matrices has on the outcome of a turmite. Using a similar program as before, we ran through every single possible 2-state, 2-colour turmite, while checking these properties for all three rule matrices: the state matrix, the colour matrix and the turn matrix. As a result, we obtained the following proportion table:

Condition	**Highway**	**Busy beaver**
State matrix invertible	45.7%	54.3%
Colour matrix invertible	50.1%	49.9%
Turn matrix invertible	38.6%	61.4%
State matrix symmetric	43.6%	56.4%
Colour matrix symmetric	42.7%	57.3%
Turn matrix symmetric	42.8%	57.2%
State matrix diagonalisable	44.0%	56.0%
Colour matrix diagonalisable	43.5%	56.5%
Turn matrix diagonalisable	42.7%	57.3%
State matrix NOT invertible	42.1%	57.9%
Colour matrix NOT invertible	39.4%	60.6%
Turn matrix NOT invertible	58.0%	42.0%
State matrix NOT symmetric	43.3%	56.7%
Colour matrix NOT symmetric	44.2%	55.8%
Turn matrix NOT symmetric	43.7%	56.3%
State matrix NOT diagonalisable	41.7%	58.3%
Colour matrix NOT diagonalisable	43.2%	56.8%
Turn matrix NOT diagonalisable	51.0%	49.0%

This table shows us the proportions of highways and busy beavers when the condition stated in the first column is verified. Overall, we notice that under most conditions the formation of busy beavers is visibly favoured. This overlaps with our previous assessment, where we found that busy beavers tend to create themselves more frequently than highways in the case of the 2-state, 2-colour turmite. From our analysis, we can see that this was especially the case when the turn matrix is invertible and when the colour matrix

is not invertible. The only conditions that seem to give preference to the formation of highways are if the colour matrix is invertible, if the turn matrix is not invertible and if the turn matrix is not diagonalisable. However, it is important to realise that this could be due to the low number of possible 2×2 colour and turn matrices that have these properties, giving us a smaller sample size to work with.

4.3 Conjectures

As a result, we have come up with the following conjectures for the overall behaviour of the 2-state, 2-colour turmite:

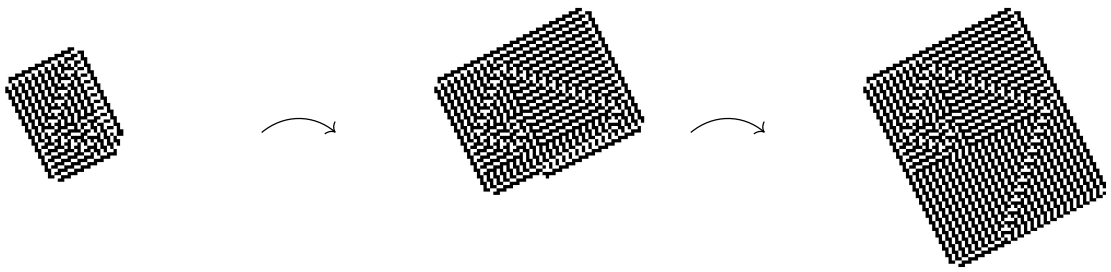
- Every single 2-state, 2-colour turmite on a blank grid will eventually either form a highway or a busy beaver, with the proportions being 43.5% and 56.5% respectively.
- Every single 2-state, 2-colour turmite on a blank grid forms a repeating cycle before 1˙000˙000 steps and before exiting a grid of size 10˙000 × 10˙000.
- In general, the invertibility, the symmetry and the diagonalisability of the three rule matrices do not seem to have a significant impact on the behaviour and outcome of the 2-state, 2-colour turmite.

5 Extraordinary shapes & patterns

Let us now present some of the more beautiful and fascinating turmites, that were simulated and captured visually using random, as well as purposefully chosen rule matrices of all sorts of sizes. Since there are infinitely many possible patterns, we had to narrow our selection down to just a few. Each example is accompanied by a short description of the specific rule matrices and by images of the eventual outcome, which we obtained from our own program.

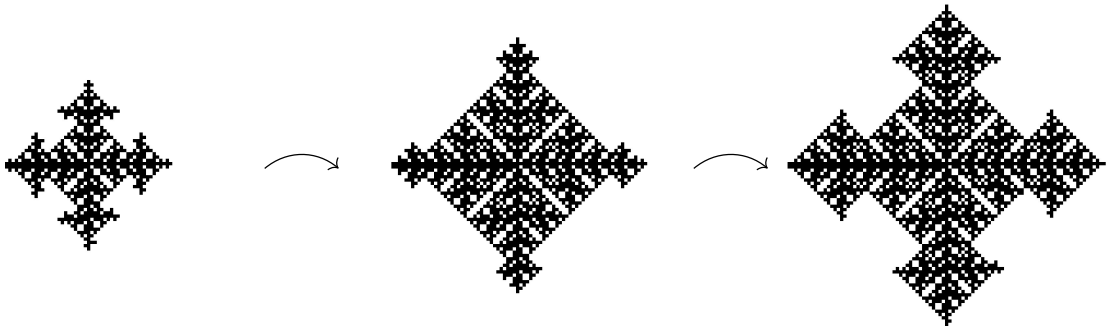
Golden spiral

First of all, the images down below represent a 2-state, 2-colour turmite after 2˙000, 5˙000 and 8˙000 steps respectively. Here, we can see that it tends to form a spiral which is very similar to the *Golden spiral*. The rules for this turmite were taken from [8].



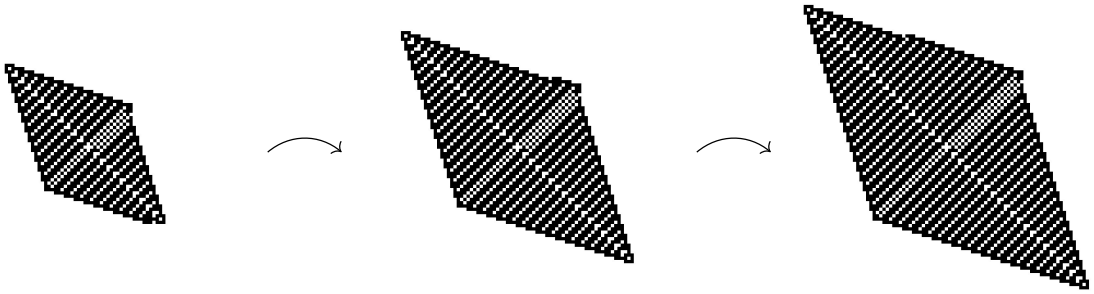
Snowflake fractal

Moreover, the images down below illustrate a 3-state, 2-colour turmite after 10˙000, 25˙000 and 40˙000 steps respectively. Here, we can observe that it tends to form a fractal which looks like a snowflake. The rules for this turmite were taken from [8].



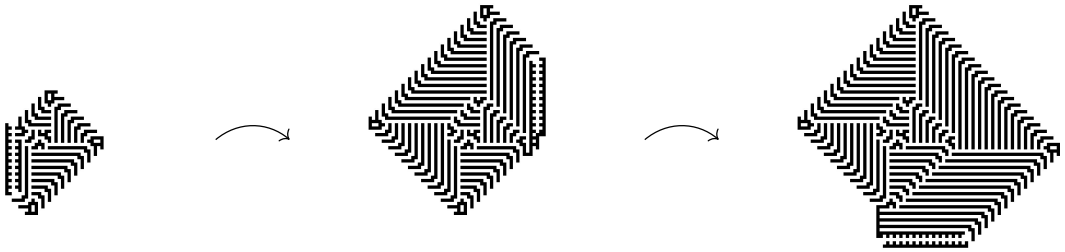
Slanted diamond

Furthermore, the images down below represent a 2-state, 2-colour turmite after 2˙000, 4˙000 and 6˙000 steps respectively. Here, we can see that it tends to form a diamond shape, slanted slightly to the left. The rules for this turmite were taken from [9].



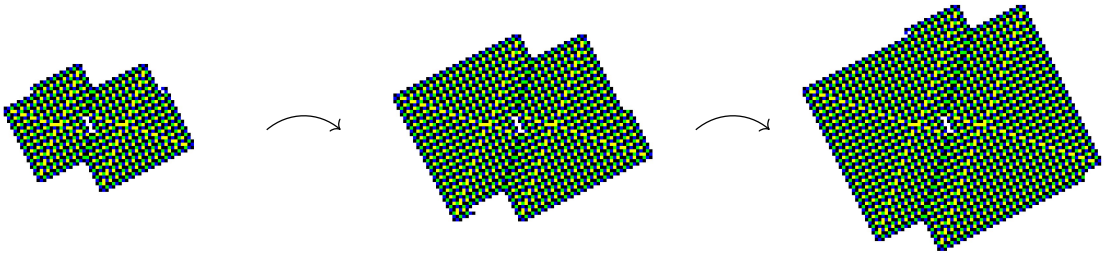
Fan spiral

In addition, the images down below illustrate a 2-state, 2-colour turmite after 2˙000, 6˙000 and 10˙000 steps respectively. Here, we can observe that it tends to form a spiral, which almost looks like the front of a fan. The rules for this turmite were taken from [9].



Two turmites symmetric spiral

Finally, we also wanted to present an example of a grid containing two turmites, defined by the same rule matrices. In general, they tended to produce very symmetrical pictures, as long as there trails did not interfere. The images down below represent two 6-state, 6-colour turmites after 1˙000, 2˙000 and 3˙000 steps respectively. Here, we were able to notice that both turmites continuously added layer after layer, while circulating the shape and complementing each other's growth. This example came up during our own experiments.



6 Conclusion

All in all, we can conclude that, even though the movement of the turmite might seem chaotic and random at first, it will generally form a periodic pattern after a certain number of steps. We were able to classify the majority of all turmites we simulated into two main categories, the highways and the busy beavers. We found that the frequency of both outcomes closely depends on several factors such as the rule matrix sizes, the total number of simulated steps, the grid size and the initial colour configuration of the grid. Moreover, we managed to determine further interesting properties like the average number of steps it takes to reach a repetitive pattern or the average cycle length, as well as the impact that the rule matrices have on the outcome proportions. Furthermore, we were able to show a weaker version of our main conjecture, which states that all turmites eventually form a highway or a busy beaver, by simulating and analysing every possible 2-state, 2-colour turmite.

The principal objective of the "Experimental mathematics" course is to give young mathematicians, like us, the opportunity to independently work on a mathematical project and gather their very first research experience. Under the guidance of Ms. Tara Trauthwein, we were able to broaden our mathematics horizon and explore unfamiliar topics. Overall, we feel like this whole experience has been very rewarding in many different ways.

Finally, we would like to encourage anybody who is fascinated by all types of cellular automata and has some knowledge of mathematical and programming concepts, to try out similar simulations and come up with their very own conjectures.

7 Appendix

The following two Python codes represent the programs we used to simulate the visual and analytical versions of the turmite. The majority of the other codes that we used were based on these two codes and were obtained by simply changing minor details.

```
import pygame
from pygame.locals import *
from random import randrange


class Square:  # here we define the class Square; each square of our grid will be an object of this class
    colours = {0: 'white', 1: 'black', 2: 'red', 3: 'blue', 4: 'yellow', 5: 'green', 6: 'pink', 7: 'orange',
               8: 'purple', 9: 'turquoise', 10: 'brown', 11: 'grey'}

    def __init__(self, x, y, size, state, colour, direction):
        self.x = x
        self.y = y
        self.size = size
        self.state = state
        self.colour = colour
        self.direction = direction

    def draw_square(self, screen):
        pygame.draw.rect(screen, Color(Square.colours[self.colour]),
                         (self.size * self.x, self.size * self.y, self.size, self.size))


class Grid:  # here we define the class Grid; in this code we only with one object of the class Grid at a time, meaning we generate a new grid each time we change our rule matrices
    def __init__(self, size, state_matrix, colour_matrix, turn_matrix, start_point):
        self.size = size
        self.squares = []
        for x in range(self.size):
            row = []
            for y in range(self.size):
                square = Square(x, y, 10, 0, 0, 0)
                if (x, y) == start_point:
                    self.turmite = square
                row.append(square)
            self.squares.append(row)
        self.visited_squares = [self.turmite]
        self.state_matrix = state_matrix
        self.colour_matrix = colour_matrix
        self.turn_matrix = turn_matrix

    def draw_grid(self, screen):
        screen.fill(Color("white"))
        for s in self.visited_squares:
            s.draw_square(screen)

    def execute(
```

```
46              self):  # this is the essential part of this code; it is
    responsible for updating the states, colours & directions and also
    moves the turmite to the next square
47          new_state, new_direction = self.state_matrix[self.turmite.state
    ][self.turmite.colour], (
48                  self.turmite.direction + self.turn_matrix[self.turmite.
    state][self.turmite.colour]) % 4
49          self.turmite.colour = self.colour_matrix[self.turmite.state][
    self.turmite.colour]
50          if new_direction == 0 and self.turmite.y > 0:
51              self.turmite = self.squares[self.turmite.x][self.turmite.y -
     1]
52              self.turmite.state = new_state
53              self.turmite.direction = new_direction
54          elif new_direction == 1 and self.turmite.x < self.size - 1:
55              self.turmite = self.squares[self.turmite.x + 1][self.turmite
    .y]
56              self.turmite.state = new_state
57              self.turmite.direction = new_direction
58          elif new_direction == 2 and self.turmite.y < self.size - 1:
59              self.turmite = self.squares[self.turmite.x][self.turmite.y +
     1]
60              self.turmite.state = new_state
61              self.turmite.direction = new_direction
62          elif new_direction == 3 and self.turmite.x > 0:
63              self.turmite = self.squares[self.turmite.x - 1][self.turmite
    .y]
64              self.turmite.state = new_state
65              self.turmite.direction = new_direction
66          else:
67              return True
68          if self.turmite not in self.visited_squares:
69              self.visited_squares.append(self.turmite)
70          return False
71
72
73  grid_size = 100
74
75  name = input('Please enter the name of your file : ')
76  states = int(input('Please enter number of states your random turmite
    should have : '))
77  colours = int(input('Please enter the number of colours your random
    turmite should have : '))
78
79  state_matrix = []
80  colour_matrix = []
81  turn_matrix = []
82
83  for i in range(states):  # here the different rule matrices are filled
    with random valid values
84      state_row, colour_row, turn_row = [], [], []
85      for j in range(colours):
86          state_row.append(randrange(states))
87          colour_row.append(randrange(colours))
88          turn_row.append(randrange(4))
89      state_matrix.append(state_row)
90      colour_matrix.append(colour_row)
91      turn_matrix.append(turn_row)
```

```
92
93 start_point = (grid_size // 2, grid_size // 2)
94
95 grid = Grid(grid_size, state_matrix, colour_matrix, turn_matrix,
       start_point)
96
97 clock = pygame.time.Clock()
98
99 pygame.init()  # here we initialise pygame; this allows us to project
       the simulation on to the screen & observe everything visually
100 size = 10 * grid_size
101 FPS = 50
102 screen = pygame.display.set_mode((size, size))
103 pygame.display.set_caption("Turmites")
104 screen.fill(Color("white"))
105
106 grid.draw_grid(screen)
107 pygame.display.update()
108 rect = pygame.Rect(0, 0, size, size)
109
110 max_steps = 100000
111 steps = 0
112 done = False
113
114 while not done:  # here the simulation begins; the execute function from
        above is called at each iteration & the loop only stops when either
       the turmite exits the bounded grid or when the maximum number of
       steps is reached
115     for event in pygame.event.get():
116         if event.type == QUIT:
117             done = True
118     done = grid.execute()
119     clock.tick(FPS)
120     grid.draw_grid(screen)
121     pygame.display.update()
122     steps += 1
123     if steps % 100 == 0 or done:
124         sub = screen.subsurface(rect)
125         screenshot = pygame.Surface((size, size))
126         screenshot.blit(sub, (0, 0))
127         pygame.image.save(screenshot, f'{name} (after {steps} steps).jpg
       ')
128     if steps == max_steps:
129         done = True
130
131 pygame.quit()
132
133 print(f'State matrix: {state_matrix}')
134 print(f'Colour matrix: {colour_matrix}')
135 print(f'Turn matrix: {turn_matrix}')
```

Listing 1: Code of visual program

```
1 from random import randrange
2 from xlwt import Workbook
3
4
5 class Square:
6     def __init__(self, x, y, size, state, colour, direction):
```

```
        self.x = x
        self.y = y
        self.size = size
        self.state = state
        self.colour = colour
        self.direction = direction


class Grid:
    def __init__(self, size, state_matrix, colour_matrix, turn_matrix, start_point):
        self.size = size
        self.squares = []
        for x in range(self.size):
            row = []
            for y in range(self.size):
                square = Square(x, y, 10, 0, 0, 0)
                if (x, y) == start_point:
                    self.turmite = square
                row.append(square)
            self.squares.append(row)
        self.visited_squares = [self.turmite]
        self.path = [(self.turmite.x, self.turmite.y, self.turmite.state, self.turmite.colour,
                      self.turmite.direction)]  # this list contains the sequence of all squares (coordinates, states, colours, directions) which the turmite has visited (with repetition)
        self.state_matrix = state_matrix
        self.colour_matrix = colour_matrix
        self.turn_matrix = turn_matrix

    def add_to_path(self):
        self.path.append(
            (self.turmite.x, self.turmite.y, self.turmite.state, self.turmite.colour, self.turmite.direction))

    def execute(self):
        new_state, new_direction = self.state_matrix[self.turmite.state][self.turmite.colour], (
                self.turmite.direction + self.turn_matrix[self.turmite.state][self.turmite.colour]) % 4
        self.turmite.colour = self.colour_matrix[self.turmite.state][self.turmite.colour]
        if new_direction == 0 and self.turmite.y > 0:
            self.turmite = self.squares[self.turmite.x][self.turmite.y - 1]
            self.turmite.state = new_state
            self.turmite.direction = new_direction
        elif new_direction == 1 and self.turmite.x < self.size - 1:
            self.turmite = self.squares[self.turmite.x + 1][self.turmite.y]
            self.turmite.state = new_state
            self.turmite.direction = new_direction
        elif new_direction == 2 and self.turmite.y < self.size - 1:
            self.turmite = self.squares[self.turmite.x][self.turmite.y + 1]
            self.turmite.state = new_state
            self.turmite.direction = new_direction
```

```
        elif new_direction == 3 and self.turmite.x > 0:
            self.turmite = self.squares[self.turmite.x - 1][self.turmite
    .y]
            self.turmite.state = new_state
            self.turmite.direction = new_direction
        else:
            return False
        if self.turmite not in self.visited_squares:
            self.visited_squares.append(self.turmite)
        self.add_to_path()
        return True


def pattern_in_path(
        path):  # this is both the most important & most complex part of
     the code; it allows us to check for repetitive beahviour of our
    turmite
     counter = 0  # this counter helps us to determine the number of
    periodic cycles in a row; here we used the convention that if 5
    cycles appear in a row then we can classify the turmite as either a
    highway or a busy beaver
     for i in range(len(path) - 2, 1,
                    -1):  # we start at the back of our path list and
    search for the most recent square, where our turmite had the same
    state, colour & direction as it did for the last square in the path
        busy_beaver, highway = True, True
        if path[i][2:] == path[-1][2:]:
            for k in range(1,
                           len(path) - i):  # after having found the
    most recent square having this property, we check if the squares
    preceding the most recent square and the squares preceding the last
    square of the path share the exact same properties
                if path[-k - 1][2:] != path[i - k][
                                       2:]:  # if they don't have the
    same properties (e.g. same colour) then the path from the i th square
     to the last square of the path doesn't represent a full cycle
                    highway = False  # so we can conclude that for this
    i, we don't have a highway, nor a busy beaver (since a busy beaver is
     a special case of a highway); however, this condition might hold
    true for an i situated earlier in the path list
                    counter = 0  # we reset our counter to 0 a break out
     of the loop
                    break
                if busy_beaver and path[-k - 1][:2] != path[i - k][
                                                      :2]:  # if the
    squares preceding don't have the exact same coordinates, then it can'
    t be a busy beaver for that particular i (however it could still
    possibly be a highway)
                    busy_beaver = False
                    # if it is neither a highway, nor a busy beaver for
    this value i, we continue repeating this whole procedure with the
    second most recent square, where our turmite had the same state,
    colour & direction as it did for the last square in the path
            if highway:  # here we check if a type of highway was formed
     (remember that a busy beaver is simply a special type of highway)
                counter += 1  # here we increment our counter by 1 since
     we have been able to detect a full cycle (either highway or busy
    beaver)
```

```
86                  if counter == 5:  # once we have detected 5 cycles in a
    row we can assume that this cycle will repeatedly form itself; this
    is the "main weakness" of our program, since we can only check for
    repetitive cycles a finite number of times
87                      return 'Busy beaver' if busy_beaver else 'Highway'
    # here we determine which type of cycle was formed
88      return 'Random movement'  # if none of the above conditions are
    verified, the movement of our turmite can be classified as random
89
90
91  grid_size = 1000
92
93  for t in range(2, 11):
94      wb = Workbook()  # here we initialise our excel workbook; this is
    essential to keep track of all the important statistics gathered
    during the simulation
95      sheet1 = wb.add_sheet('Sheet 1')
96      sheet1.write(0, 0, 'Total # of simulations')
97      sheet1.write(0, 1, 'Random movement')
98      sheet1.write(0, 2, 'Highway')
99      sheet1.write(0, 3, 'Busy beaver')
100
101     outcome = {'Random movement': 0, 'Highway': 0, 'Busy beaver': 0}
102     max_steps = 100000
103
104     for a in range(1, 10001):
105         state_matrix = []
106         colour_matrix = []
107         turn_matrix = []
108
109         for i in range(t):
110             state_row, colour_row, turn_row = [], [], []
111             for j in range(t):
112                 state_row.append(randrange(t))
113                 colour_row.append(randrange(t))
114                 turn_row.append(randrange(4))
115             state_matrix.append(state_row)
116             colour_matrix.append(colour_row)
117             turn_matrix.append(turn_row)
118
119         start_point = (grid_size // 2, grid_size // 2)
120
121         grid = Grid(grid_size, state_matrix, colour_matrix, turn_matrix,
    start_point)
122
123         result = 'Random movement'
124
125         for n in range(max_steps):
126             if grid.execute():
127                 if n % 1000 == 0:  # every 1000 steps we check to see if
    any repetitive pattern can be spotted over the last 1000 steps
128                     result = pattern_in_path(grid.path[n - 1000:])
129                     if result != 'Random movement':
130                         break
131             else:
132                 result = pattern_in_path(grid.path[max(0, n - 1000):])
133                 break
134
```

```
        outcome[result] += 1

        print(a)
        print(result)

    sheet1.write(1, 0, 10000)
    sheet1.write(1, 1, outcome['Random movement'])
    sheet1.write(1, 2, outcome['Highway'])
    sheet1.write(1, 3, outcome['Busy beaver'])

    wb.save(f'Turmites outcome stats for {t}-state, {t}-colour random
matrices (after {max_steps} steps).xls')
```

Listing 2: Code of analytical program

As we are unable to showcase all of the data collected during our research period, we have decided to upload all of our results and Python code to an online folder, which is linked down below:

https://drive.google.com/drive/folders/1_zZDBcI5RjqOhsomJOxV_20BYFD9RYOv?usp=sharing

References

[1] Brady, Allen H., 1988, *The Busy Beaver Game and the Meaning of Life*, Springer-Verlag

[2] Gajardo, A., Goles, E., Moreira, A., 2002, *Complexity of Langton's ant*, Discrete Applied mathematics

[3] Gajardo, A., Goles, E., Moreira, A., 2001, *Generalized Langton's Ant: Dynamical Behavior and Complexity*, Springer-Verlag

[4] Langton, Chris G., 1986, *Studying artificial life with cellular automata*, Physica D: Nonlinear Phenomena

[5] Troubetzkoy, S., 1997, *The Lewis-Parker lecture: The ant*, Alabama J. Mathematics

[6] Wikipedia, 2022, *Conway's Game of Life*, viewed 13 May 2022, `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`

[7] Wikipedia, 2022, *Langton's ant*, viewed 13 May 2022, `https://en.wikipedia.org/wiki/Langton%27s_ant`

[8] Wikipedia, 2022, *Turmite*, viewed 13 May 2022, `https://en.wikipedia.org/wiki/Turmite`

[9] Youtube, 2022, *Turmite Animations*, viewed 24 May 2022, `https://www.youtube.com/watch?v=8NIaddAGdJA`