

Game of Life

Non-Examined Assessment

Jonathan Maud

Table of Contents

Introduction	5
Notes from Interview.....	5
What is the Proposal?.....	5
Initial Research	6
Forms of Teaching Methods.....	6
John Horton Conway's Game of Life.....	9
Conclusion.....	10
Potential User.....	11
What my Program should include?.....	11
Problems when programing the project.....	11
Solution.....	11
Background of Problem.....	11
Flowchart of Single Cell:.....	11
The Simulation.....	11
Data Storage.....	12
Survey.....	12
What can be added to my project?.....	12
Requirements	14
Success Criteria.....	14
Time Plan.....	15
Design	16
User Interface.....	16
Flowchart showing interactivity:.....	18
Flow Charts:.....	20
Generate Population Flowchart.....	20
Rules of the Game Flowchart:.....	21
IPSO Chart:.....	22
UML Class Diagram:.....	22
Pseudo code.....	24
High Level Flowchart.....	26
Implementation	27
Screen Shots of the Implementation Process.....	27
Testing orientation of x and y coordinates.....	27
Correcting the x and y coordinates in early testing.....	27
First working prototype.....	33

Changing the Visual Speed Output	35
Moving the project to Forms:	37
Input Method	41
Adding this to my solution	42
Simplifies Output	45
Integrating the Input Routine	47
Finished Base Code	49
Technical Solution	56
Prototyping the Stop Button by using timer function	59
Adding Functionality to Exit Button	59
Speed Variations	61
File Handling Prototype	62
Reset Button	65
Testing for robustness with exceptional data.....	66
Intermediate Stage Feedback	68
Prototyping the Stamp Function.....	69
Prototyping the Stamp Tool.....	70
Presets.....	71
Adding New Preset 2.....	73
Adding New Preset 3.....	74
Adding New Preset 4.....	75
Adding New Preset 5.....	76
Adding New Preset 6.....	77
Object Orientated Code	78
End User Testing	83
Testing	88
GUI Testing Table	88
Beta Testing	93
Beta Testing Result	94
Evalutaion	95
What went well:.....	95
End User Sign Off	95
Feedback/ Evidence	95
Evaluation of Success Criteria	96
What I would change if I was to do this again:	101
Future development	101

Conclusion..... 102
Appendix: Full Code Listing..... 103

Introduction

In order to do this project, I began by asking my friends and family if they had any problems suitable for me to investigate and possibly solve. However, none of them were very helpful and so I expanded my search. Eventually my brother Sebastian Maud told me of a project that he thought I might be interested in. Sebastian attends my old school and is currently in year 11.

Sebastian told me that his biology teacher has been speaking to him about the fact that his GCSE students were finding it hard to understand how environmental factors might affect the growth or decline of an organism. I felt that I might be able to help and so I arranged to have an initial meeting with Mr Watson so that I could clarify the problem. In order to help me I took notes displayed below.

Notes from Interview-

Me: "Thank You for seeing me Mr Watson, I believe there is a problem you discussed with my brother"

Mr Watson: "Yes. My Year 11 Students are finding it difficult to visualise how environmental effects can shape a collection of organisms over time. My top students have it, but I have a number of visual learners in my class who I would like to engage with more fully. I used to have a simulation on VHS video, but since the upgrade in systems my VHS player no longer can connect with the new system."

Me: "Could you describe the contents of the video?"

Mr Watson: "Sure, it was very good. It consists of time lapses showing the growth of bacteria on a pond. In the first instance the environmental factors are in favour of the bacteria to too greater level, and the bacteria chokes the ecosystem. In the second instance a retardant is added to the surface of the pond halfway through, this inhibits the growth of the bacteria and therefore allows other life forms to flourish. In this case the growth of the bacteria was slow enough for other pond life to feed on it before choked the pond and the bacteria disappeared. In the third instance the environmental conditions inhibited growth to just the right amount to provide a balance in the ecology, and this meant that both the bacteria and the pond life survived."

Me: "I am doing a project and I think I might be able to model something similar on a computer program would that be of interest to you?"

Mr Watson: "Yes, that would be great."

Me: "Ok, I will look at possible solutions for you, and get back to you."

I felt that this problem had the potential to be a suitable project, and so I investigated it further.

What is the Proposal?

My Project will be used to help teach the growth of bacteria to students. I will create a program that will simulate the growth of cells abiding by the rules that the teacher has decided. Presently, there are very few simulations that exist with the same rules, but don't give anywhere near the amount of detail. The Year 11 students will be able to visually study the growth of cells in a certain area. The simulation will give the students a much more interactive visual way to help the students that prefer to learn in that way. When the program is completed Mr Watson should be able to teach the topic more easily and that will lead to improved grades within his classes.

Initial Research

Forms of Teaching Methods

The first thing I did was to see how this problem has been solved before. (i.e. to help students understand growth and decline in an environmental setting.)

1- BBC Bitesize discuss reproduction but only on a cellular level. They did not consider multiple organisms and how these may interact.

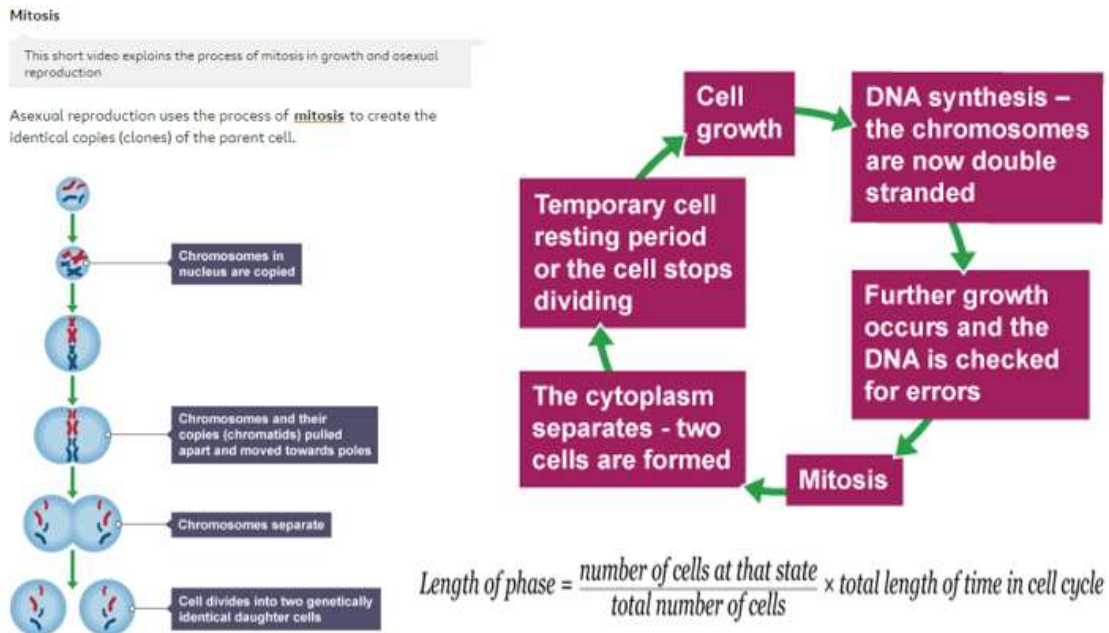


Figure 1: Reproduction as explained by Bitesize

2-Crash Course (YouTube Channel) this channel specialises in providing video for school children on various topics. The content is good and relevant however there are no animations and so would not appeal to Mr Watson's students.

CrashCourse
10M subscribers

Crash Course

<https://www.youtube.com/watch?v=izRvPaAWgyw&list=PL3EED4C1D684D3ADF&index=40>

3-Role Play I researched other methods of teaching this topic and I understood that a method sometimes used inside the classroom is role-play. In this method members of the class are assigned various roles or personas which is reflective of the topic being explained. Students then interact in a way that mimic the topic being taught. This will be very good for end user's students visual learning, however I have discovered a number of limitations. Number one: most of the resources on the internet for this type of activity are aimed at primary school children and therefore would be unlikely to engage a year 11 student. Number two: the type of knowledge that Mr Watson was describing related to large scale patterns comprised of tiny organisms. Such patterns would require thousands or millions of components and therefore would be hard to demonstrate using a handful of students.

4-Use of TextBooks

Copyrighted Material

B1.2 Animal and plant cells

Learning objectives
After this topic, you should know:

- the main parts of animal cells
- the similarities and differences between plant and animal cells.

Synoptic link
You will find out more about classifying the living world in Chapter B15.

Go further
The ultrastructure of a cell – the details you can see under an electron microscope – includes structures such as the cytoskeleton, the Golgi apparatus, and the rough and smooth endoplasmic reticulum. They support and move the cell, modify and package proteins and lipids, and produce the chemicals that control the way your body works.

Study tip
Learn the parts of the cells shown on these diagrams, and their functions.

Synoptic link
For more information on photosynthesis, look at Topic B8.1.

The cells that make up your body are typical animal cells. All cells have some features in common. You can see these features clearly in animal cells.

Animal cells – structure and function
The structure and functions of the parts that make up a cell have been made clear by the electron microscope (Figure 1). You will learn more about how their structure relates to their functions as you study more about specific organ systems during your GCSE Biology course. An average animal cell is around 10–30 µm long (so it would take 100 000–300 000 cells to line up along the length of a metre ruler). Human beings are animals so human cells are just like most other animal cells and you will see exactly the same structures inside them.

- The **nucleus** – controls all the activities of the cell and is surrounded by the nuclear membrane. It contains the genes on the chromosomes that carry the instructions for making the proteins needed to build new cells or new organisms. The average diameter is around 10 µm.
- The **cytoplasm** – a liquid gel in which the organelles are suspended and where most of the chemical reactions needed for life take place.
- The **cell membrane** – controls the passage of substances such as glucose and mineral ions into the cell. It also controls the movement of substances such as urea or hormones out of the cell.
- The **mitochondria** – structures in the cytoplasm where aerobic respiration takes place, releasing energy for the cell. They are very small, 1–2 µm in length and only 0.2–0.7 µm in diameter.
- The **ribosomes** – where protein synthesis takes place, making all the proteins needed in the cell.

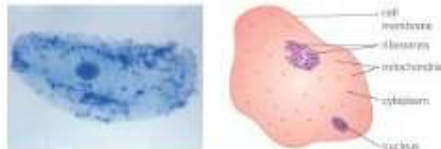


Figure 1 Diagrams of cells are much easier to understand than the real thing seen under a microscope. This picture shows a simple animal cell magnified $\times 1350$ times under a light microscope. This is the way a model animal cell is shown to show the main features common to most living cells.

Plant cells – structure and function
Plants are very different organisms from animals. They make their own food by photosynthesis. They do not move their whole bodies about from one place to another. Plant cells are often rather bigger than animal cells – they range from 10 to 100 µm in length.

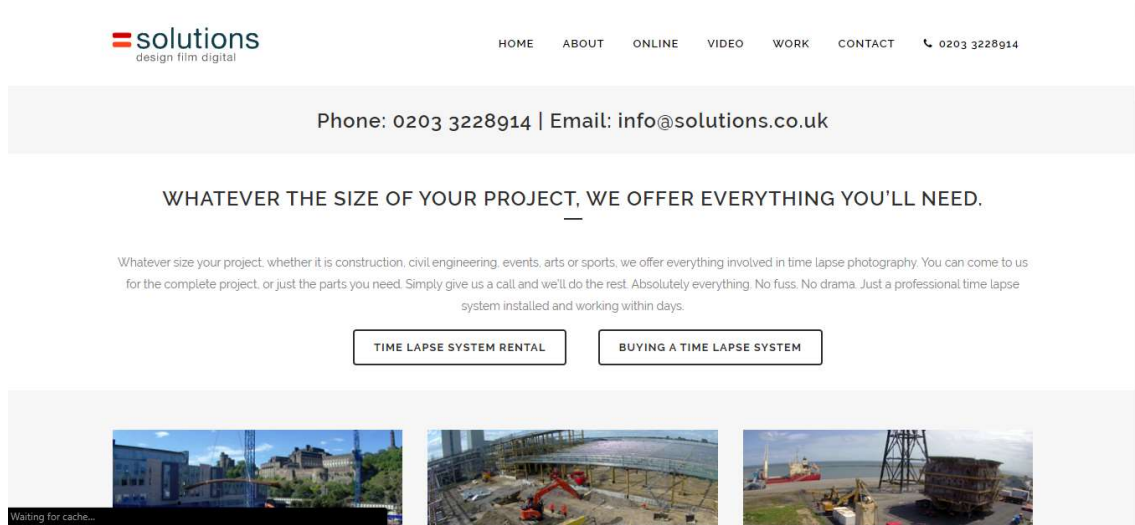
Copyrighted Material

Figure 2: AQA GCSE Biology Student Book Ann Fullick

I surveyed all of the popular GCSE biology textbooks. The closest thing that I found to what Mr Watson described is shown in figure 2. As can be seen from the screen shot the mechanism of

mitosis is clearly described using key words, but a visual learner may find it difficult to understand or remember the information.

5 The Internet -Next I searched the internet for resources that might help me solve this problem. I came across this company that specialise in creating time-lapse photography, but when I checked their prices they started at £1000 which is far too high for an educational establishment.



https://www.solutions.co.uk/video-services/time-lapse-photography/?gclid=CjwKCAjwq4fsBRBnEiwANTahcGIDb16gZTxsRrBjb27een8bUGIWz3C74HigMDTbSxJaUQCZ-nb7RoCLIsQAvD_BwE

6-Existing Computer Programs. Next I searched for a computer program that would simulate bacteria growth. The best result that I found is shown in figure 3. This model was developed by the University of Birmingham. I was very impressed with the visual interface and this seemed to be exactly what my end user needed. However, I felt that the interface was too complex for my end user, and the program would require complex configuration and input.

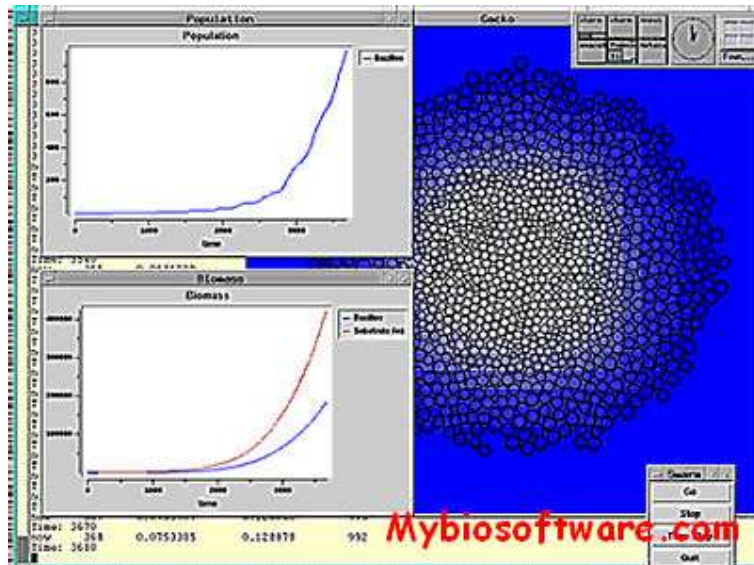


Figure 3

<http://www.mybiosoftware.com/bacsim-simulator-bacterial-growth.html>

John Horton Conway's Game of Life

As I was looking at modern programs which simulated bacterial growth I came across a reference to John Conway's Game of Life. Conway discovered 'The Game of Life' in 1970. It works by applying a few simple rules to a series of cells, and then stepping the solution through a number of iterations or 'generations'. The results were a pattern that seemed to have a life of its own, hence the name. The rules of 'The Game of Life' are as follows:

1. If the cell has less than two neighbouring cells it will die (from loneliness)
2. If the cell has two or three neighbours it carries on living to the next generation
3. If the cell has more than three cells neighbouring it the cell will die (from overcrowding)
4. If a space has exactly three neighbours then it will be come to life

What Conway discovered has been studied ever since. Evolving patterns, not obvious from initial conditions. Some patterns would die out, some patterns would find an equilibrium, and some would remain unstable. Over time, elements within the game have been identified and named. Some of these are shown below

Examples of Patterns

Still Life					
Oscillators					
Spaceship					

Example of very complex life like patterns which are created from these simple patterns can be found on You Tube, for example

<https://www.youtube.com/watch?v=C2vgICfQawE>

Conclusion

Throughout my research I came across a lot of resources that would be useful to a year 11 biology student. I was impressed by the clarity of diagram of BBC Bitesize. The book by Anne Fullick was full of keywords and written descriptions. The YouTube video was good but probably better for a student who preferred auditory learning. Finally, the computer program had excellent graphics but was too hard for GCSE level.

I found John Horton Conway's Game of Life very interesting as it displays the generation of cells using only four simple rules. Biological systems often have simple rules affecting growth, such as phototropism and geotropism, and yet from those simple rules complexity can develop. I felt that 'The Game of Life' therefore would offer an excellent basis from which to develop a computerised solution to my problem. I therefore decided to use this as a basis for my project and to include all the best features from the other resources that I have reviewed into a single computer program.

My Game of Life project aims to help Mr Watson to teach the regeneration of cells to year 11 students studying GCSE Biology. Some of his students find it difficult to learn using the currently available resources. My interactive simulation should increase engagement, allowing these students to be able to learn by visual and kinaesthetic means.

Potential User

My project is targeted specifically at Mr Watson (see Introduction) as well as any other Biology teacher that would like a visual and interactive teaching aid that demonstrates the regeneration of cells.

What my Program should include?

My project is going to be used inside the classroom and so needs to be visually appealing as well as simple to use so that it can retain students' attention. It should allow interactivity, which will allow students to feel more part of the process and should improve learner engagement.

The interactivity should allow students to be able to choose where to place the cells and then observe the growth or decline of the organism. The idea is that a few simple rules applied to a random system can produce complex outcomes.

Problems when programing the project

- The visual simulation
- Accurate interactive
- Being Visually intriguing

Solution

My project will interact with the student and teacher and allow input via an intuitive user interface. Ther should be no noticeable delay time so that the class' attention does not become distracted.

Background of Problem

My brother's Biology teacher has been speaking to him about the fact that his GCSE students where finding it hard to understand how environmental factors might affect the growth or decline of an organism and so I needed to create a simulation that displays the growth and decay of cells.

Flowchart of Single Cell:



The Simulation

My project has to simulate the topic being taught in the classroom. It is essential that it is usable and useful for the year 11 students in the classroom. It should simulate the life cycle of a cell. It

should display how the cells grow over time in an animation. The simulation should be interactive and have rapid response in order to facilitate understanding.

Data Storage

One of the features for my project is that the students will be able to save, store and load pre prepared patterns. I will use stream writer to save and load data between the program and a text file.

Survey

During my visit to Mr Watson's class I performed a piloted study. I performed a group interview of five of his students to help me with my design. I have selected some of the group answers below.

How useful would my project be to your learning experience?

Student: "It will give me a mental break from all the reading we have to do in lessons while still learning the topics at hand"

Student: "It will help a lot as I am very much a visual and physical learner so your program will allow me to be able to learn with more ease"

What Key aspects of the code must the program have?

Student: "It must be easy to use"

Mr Watson: "It must be able to engage the class in order to maximise its use"

Is there anything else that you would like to add?

Student: "No, just make the program work so it will aid me in getting a higher grade"

Student: "Make it easy to use"

The main points to take away from this:

- Importance of ease of use
- Emphasis on being aesthetically pleasing
- Needs to be animated

What can be added to my project?

In order to make my project more flexible and easier to use, I will need to add some extra features to my user interface:

- A set of instructions so that the end user (Mr Watson) and students know how to use the program
- The ability to speed up and slow down an animation so that simulations can either be played slowly to allow narration to take place, or quickly to show how generations change over time. This will also allow the visualisation to be fine tuned to run at an optimum speed on machined with different processing speeds.

- Speed control would also place students more in control of the simulation and encourage interactivity. This will encourage closer inspection benefitting learning and allow each learner experience the program to their individual needs.
- Pre created shapes/ designs that can be loaded so that a novice could get going with the software without any fore-knowledge of the rules, and also to allow more complex shapes to be loaded quickly.

Requirements

This project must be a helpful device for my end user, teacher and students in year 11 and must be created to a high standard in order for it to be worthwhile as an effective teaching device. The program will initially be based on the John Conway rules, but if time allows, other rules may be explored. The program must be robust and reliable as it will be working in an educational setting.

Success Criteria

In this section is a list of requirements that my project should fulfil will have been derived from my analysis section.

- 1) *My project's user interface must be welcoming and easy to use in the classroom environment.*
- 2) *My Solution should mimic the life cycle of cells in an artificial habitat, using the rules from Conway's Game of Life.* I will measure this criteria using feedback from the year 11 students and Mr Watson:
- 3) *My solution should be complete, self contained and fully functioning.*
- 4) *The solution that I provide must be better than, or at least complementary to, other teaching methods* (Based on feedback from the teacher Mr Watson and the year 11 students.
 - a. Interactive
 - b. Simple to use
 - c. Must be aesthetically pleasing to maximise the student's attention span
 - d. Teaches the required topic with ease
- 5) *My project must be able to store and retrieve student designs.*
- 6) *My solution must be robust and should not crash when being used* as this could disrupt lessons, resulting in wasted time and ruining the classroom atmosphere. In order to measure this, I will ask my end user (Mr Watson) to do some preliminary testing and also to keep a reliability log once my project has been given to him.
- 7) *My solution should be fully documented which will facilitate further development either myself or by a 3rd party should this be required in the future.* I will test these criteria by showing my solution to other fellow Computer Science students to see if they can easily understand my project without an explanation from myself.
- 8) *My solution should engage students*, I will judge these criteria based on the feedback from my end user (Mr Watson) and the Year 11 students.
 - a. 'Students will be able to interact using the mouse, which should focus a student's attention as they create new patterns and life forms. This freedom will encourage self learning.
 - b. The program may be seen as relaxing fun during the biology lesson, which is usually full of reading and learning from the textbooks. This program will allow a break from that cycle while still having an educational subtext.
 - c. The colours that I chose should also help make the students more engaged with my project as it will be more aesthetically pleasing, drawing and retaining a student's attention.
- 9) *My solution must be easy to use.* I will judge this criteria based on feedback from my end user (Mr Watson) and the Year 11 students.
- 10) *My solution should take up a limited amount of space on a hard drive. I am aiming for a maximum footprint of 30mb.*

- 11) *My project must be capable of running on all of the computers used at the school.*
- 12) *My solution should quick to load, I would estimate the longest time the end user (Mr Watson) and the students will have to wait would be maximum 10 seconds, based on normal attention span.*
- 13) *My project should generate an animation showing cell growth and decline.*
- 14) *My project should store the designs created by the user,* so that the student will be able to carry on where they left off last time they used the project, or so that more complex designs can be loaded quickly
- 15) *My solution must run quickly enough to create persistence of vision.*
- 16) *My solution must be capable of running on the minimum specification* computer that my end user (Mr Watson) might encounter.
- 17) *My solution must be capable of running on the school's network,* due to it having a numerous number of security protocols.

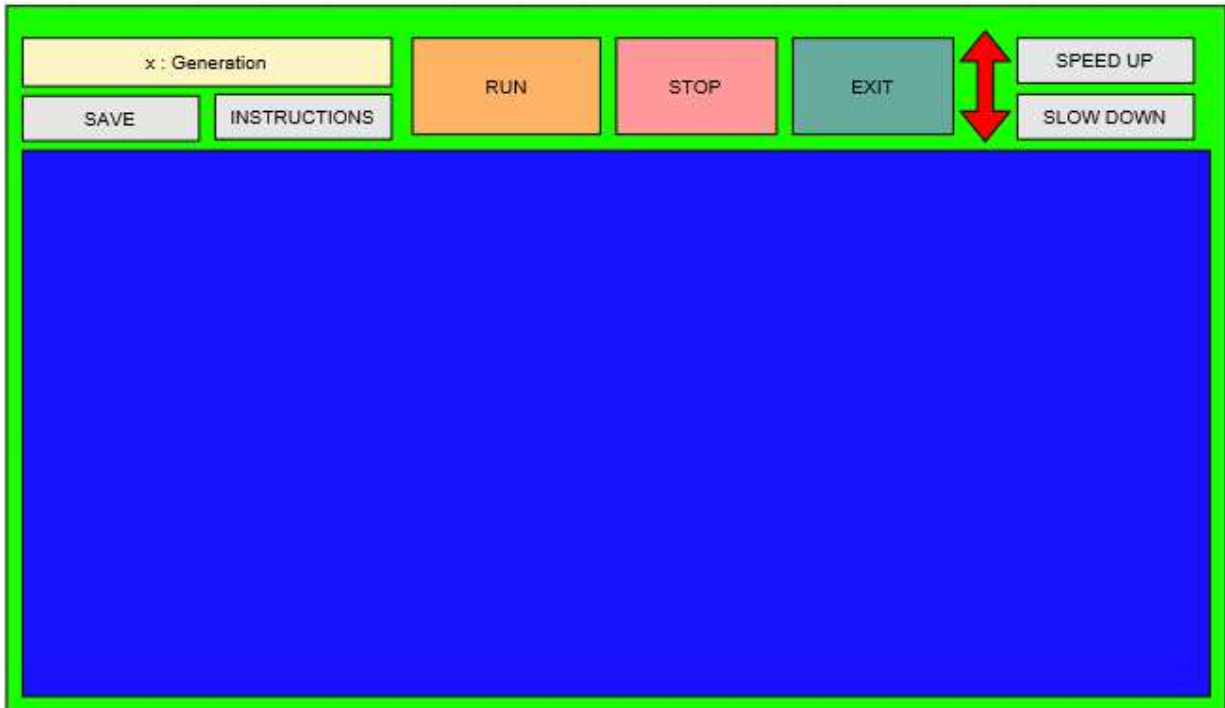
Time Plan

1. I will conduct my initial interview with my end user (Mr Watson) by 13/10/19.
2. I will complete my initial designs and develop some prototypes to return to my end user (Mr Watson) by 13/11/19.
3. I will complete my final designs by 27/11/2019.
4. I will complete the majority of my development by 17/12/2019.
5. I will aim to meet with my end user (Mr Watson) by 11/01/2020 to present my solution.
6. I will allow 2 weeks' contingency from 11/01/2020 for any changes to my project.
7. I will aim for final sign off by the end of January 2020.
8. I will aim to complete my project and submit it by the end of February 2020.
9. I will allow again allow 2 weeks' contingency from the end of February 2020 for any changes or correction my teacher may request.

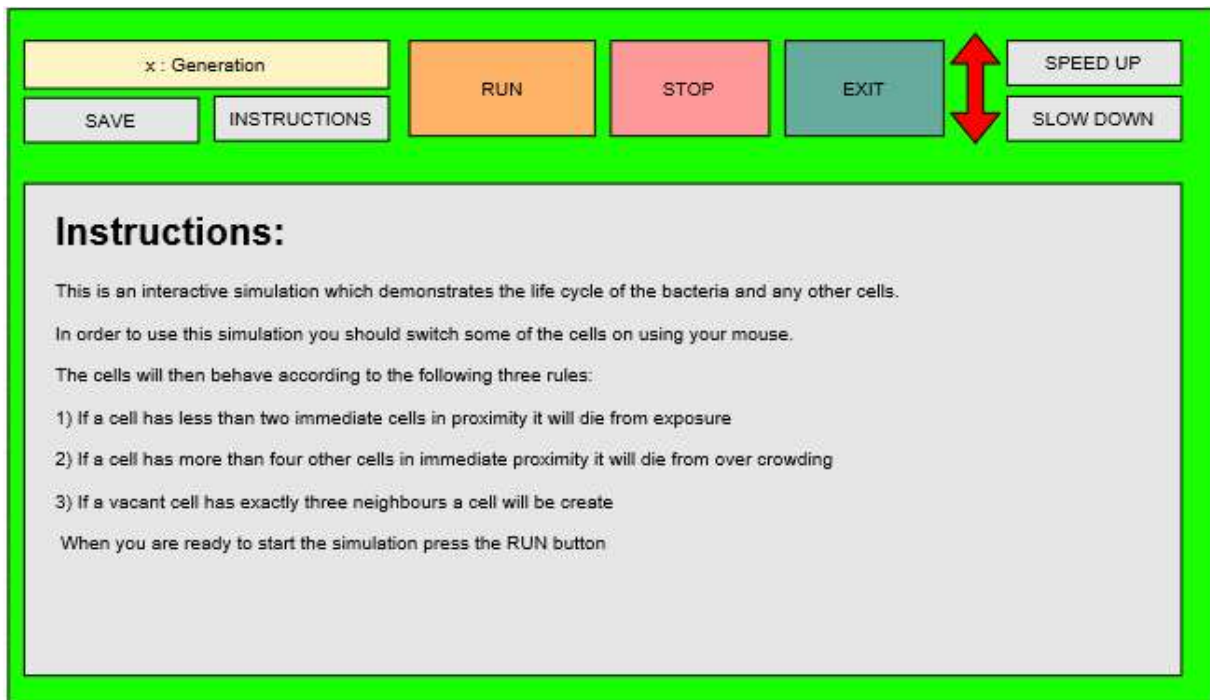
Design

User Interface

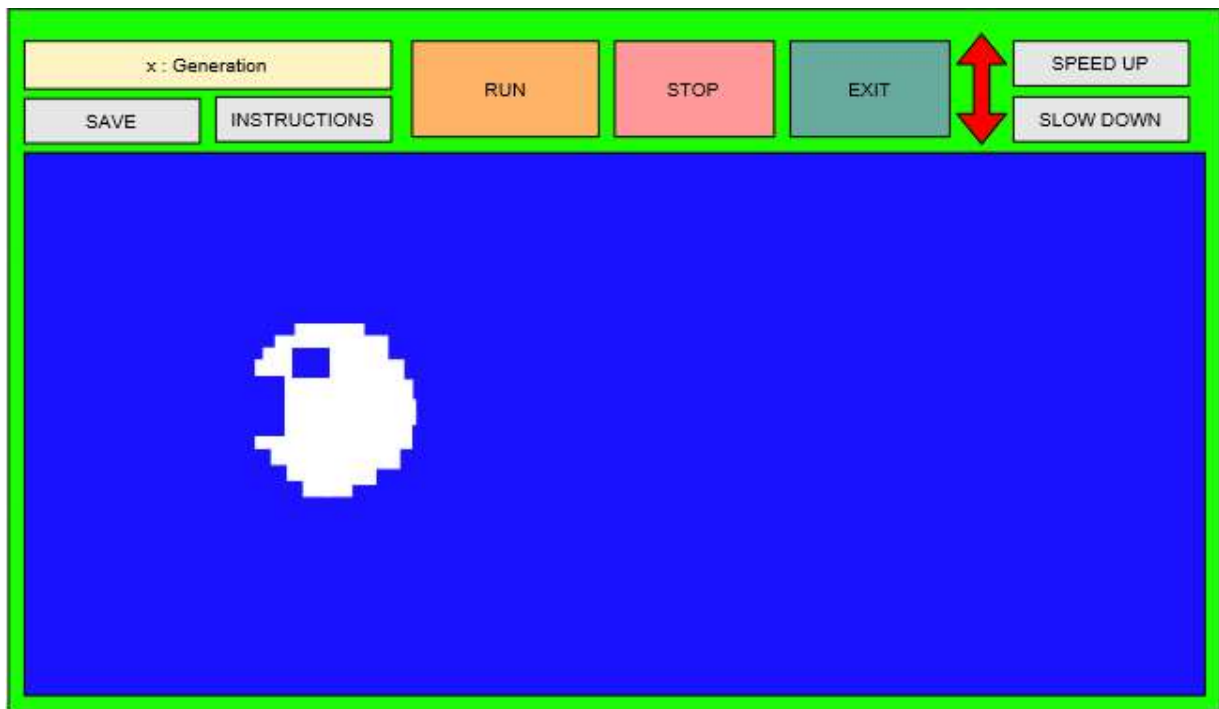
Here is what my user interface will look like:



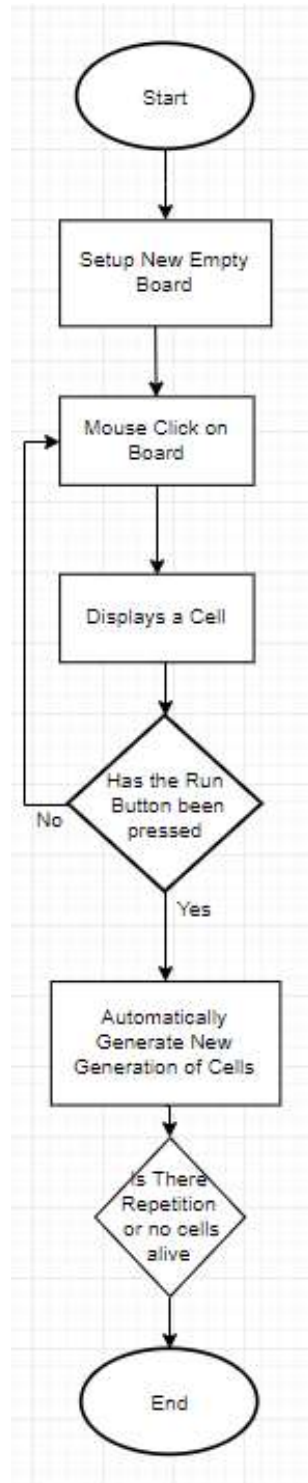
Here is the instruction screen:



Here is what my user interface will look like when it is running.



Flowchart showing interactivity:



Pseudo Code Interactive:***Pseudo Code Display:***

On-click

X = Mouseclick.X

Y = Mouseclick.y

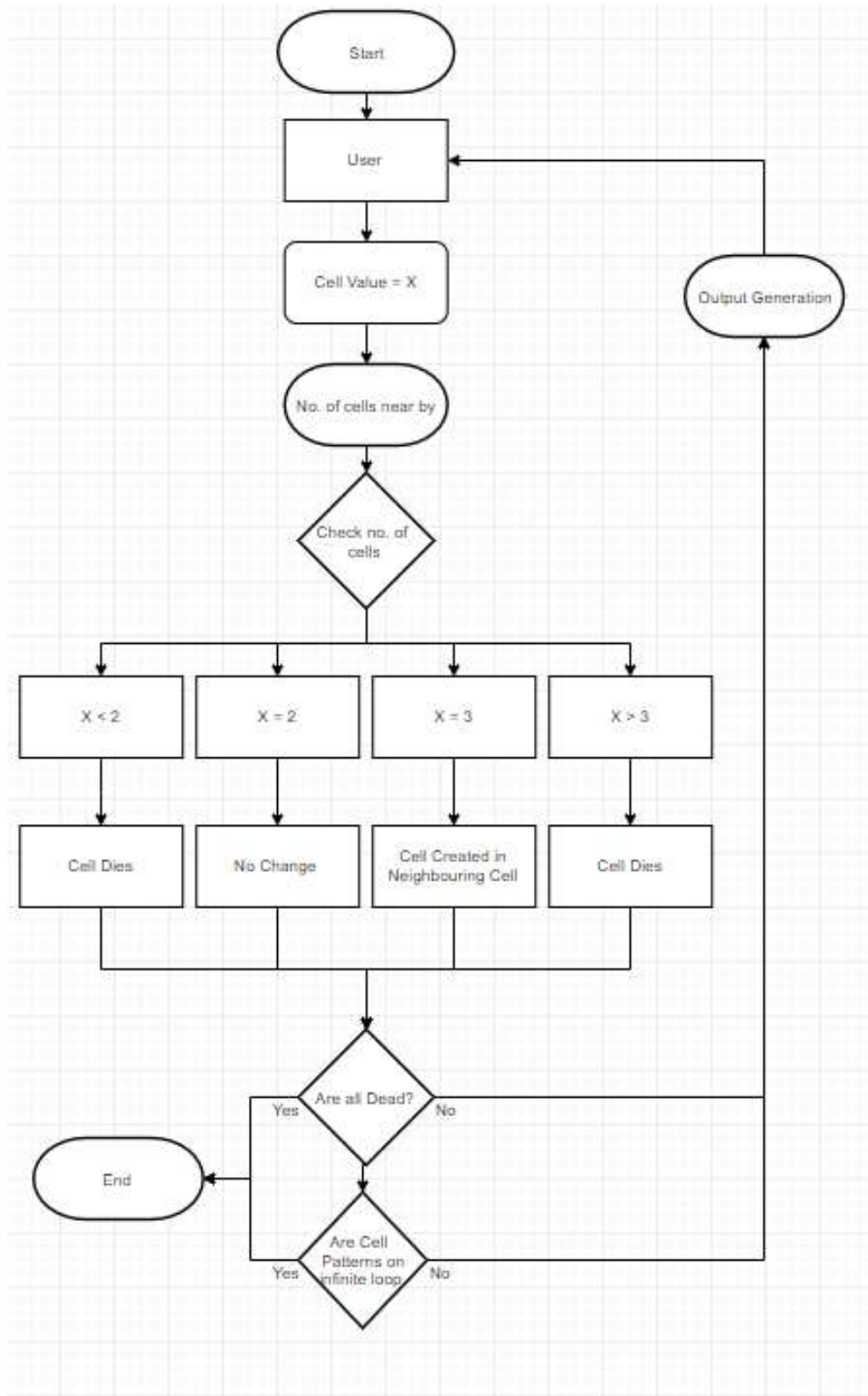
Cells.add (New Cell (X, Y))

Board Setup

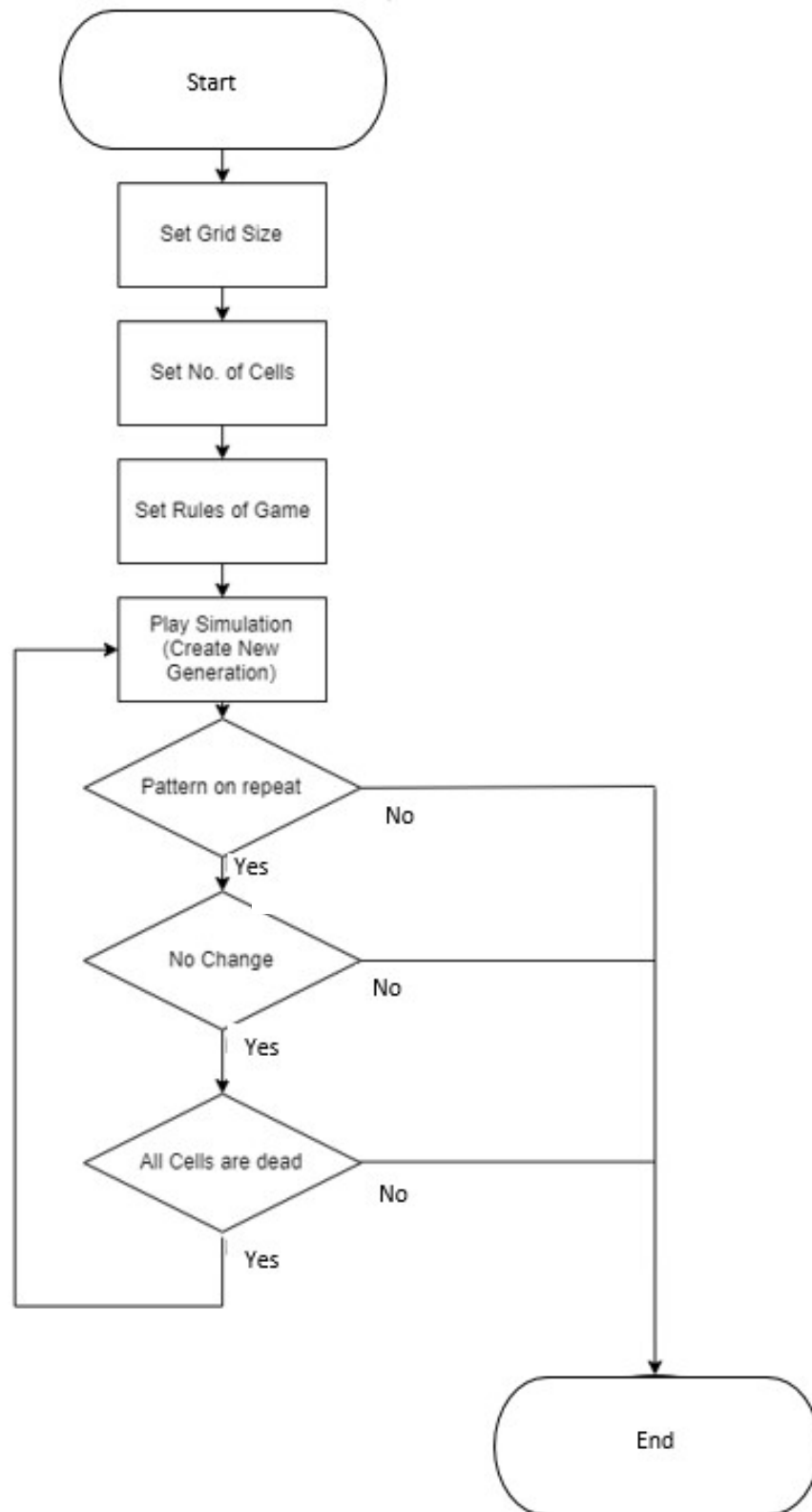
Do until Mouse Click Play

Flow Charts:

Generate Population Flowchart



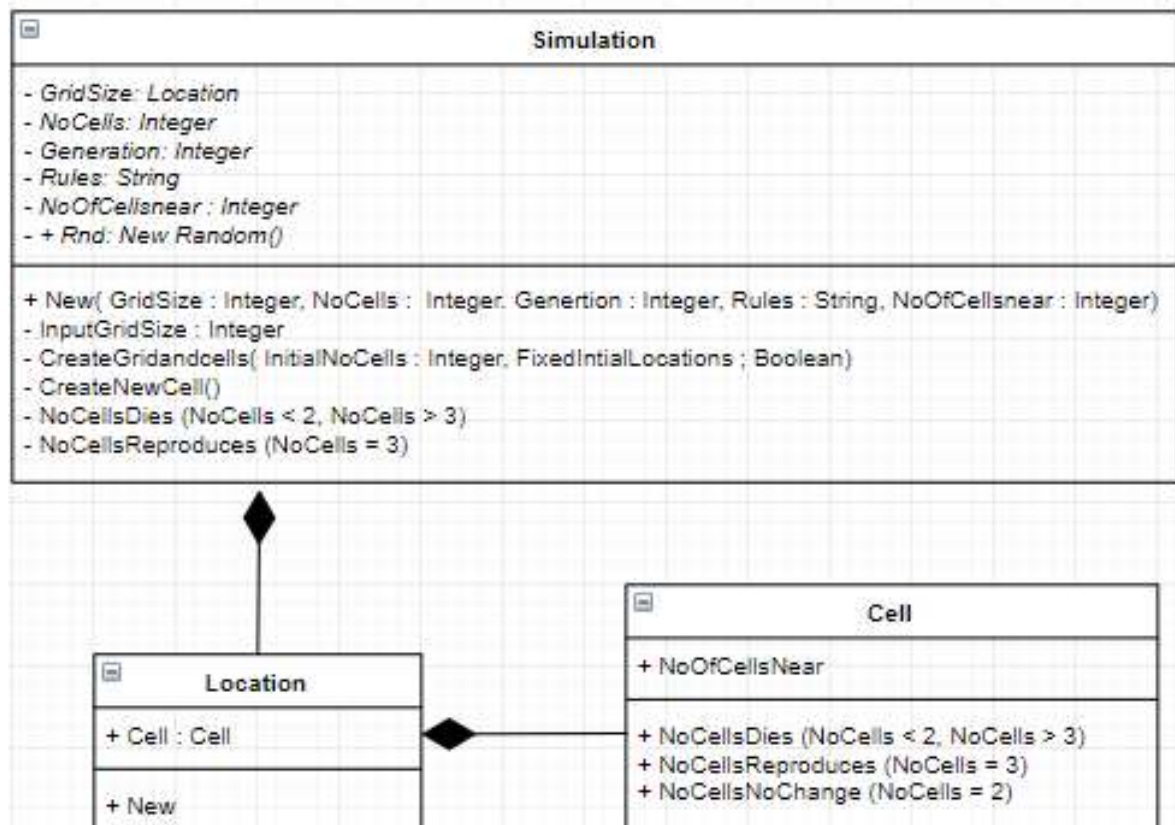
Rules of the Game Flowchart:



IPSO Chart:

<i>Inputs</i>	<i>Processes</i>
-Grid Size -Number of Cells -Game Rules	-Check Number of Cells Surrounding a Cell -Delete Cell -Create New Cell
<i>Storage</i>	<i>Outputs</i>
-Generation -Number of Cells	-Display -Board

UML Class Diagram:



Pseudo code

In this section I will discuss the main sections that I will need in my program.

Class Definition

Class Simulation

Public Definitions

Pond: String

MaximumcoordX: Integer(50)

MaximumcoordY: Integer (50)

Private Definitions (button1)

Outputline: string

Neighbours(x, y): as integer

Nextgen(x, y): as string

Generation: integer = 1

Blocksize: integer = 9

Colour: New

Private Definitions (Display)

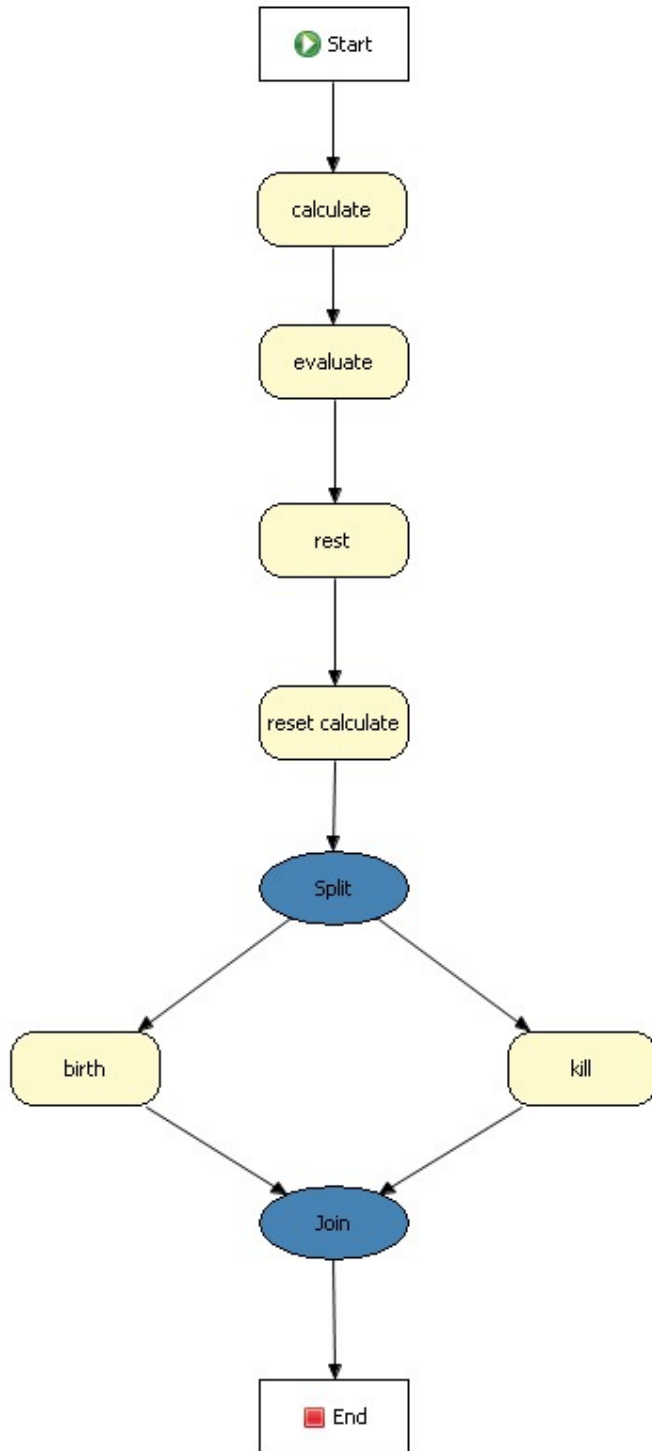
a: integer

b: integer

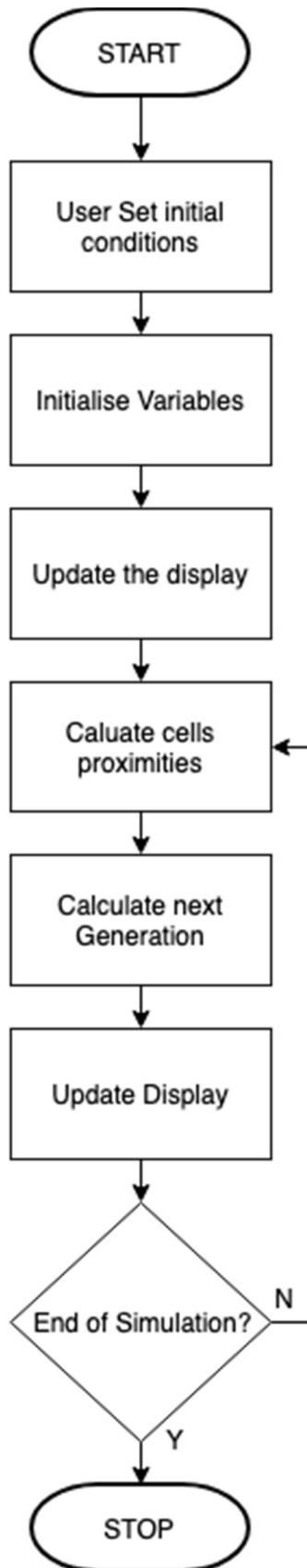
blocksize: integer = 9

Counterx, country: integer

End Class



High Level Flowchart



Implementation

Screen Shots of the Implementation Process

I began in console mode with the idea that once I solved the problem I would improve the graphical interface by making a version using forms.

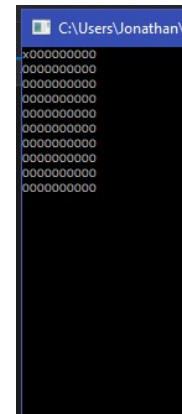
Testing orientation of x and y coordinates

```

Sub Main()
    'initialising variables
    Dim outputline As String
    Dim pond(11, 11) As String
    Dim generation As Integer = 1
    For counterx = 1 To 10
        For country = 1 To 10
            pond(counterx, country) = "0"
            neighbours(counterx, country) = 0
        Next
    Next

    For counterx = 1 To 10
        For country = 1 To 10
            pond(counterx, country) = nextgen(counterx, country)
            neighbours(counterx, country) = 0
        Next
    Next

```



Correcting the x and y coordinates in early testing.

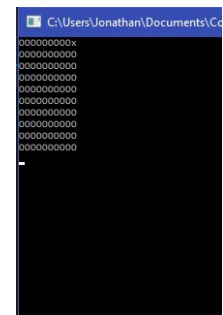
```

Sub Main()
    Dim outputline As String
    Dim pond(11, 11) As String
    Dim generation As Integer = 1
    For counterx = 1 To 10
        For country = 1 To 10
            pond(counterx, country) = "0"
            neighbours(counterx, country) = 0
        Next
    Next

    For counterx = 1 To 10
        For country = 1 To 10
            outputline = outputline & neighbours(counterx, country)
        Next
        Console.WriteLine(outputline)
        outputline = ""
    Next

    For counterx = 1 To 10
        For country = 1 To 10
            pond(counterx, country) = nextgen(counterx, country)
            neighbours(counterx, country) = 0
        Next
    Next

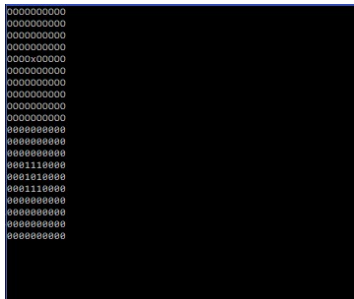
```




```

For counterx = 1 To 10
  For country = 1 To 10
    If pond((counterx - 1), (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx - 1), country) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx - 1), (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond(counterx, (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond(counterx, (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), country) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
  Next
Next

```

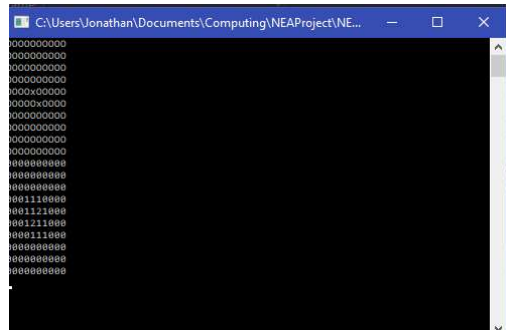


^^Solved, wrong duplicate test

```

For counterx = 1 To 10
  For country = 1 To 10
    pond(counterx, country) = "0"
    neighbours(counterx, country) = 0
  Next
Next

```



^^Testing counting the neighbouring algorithm.

```
pond(5, 5) = "x"
pond(6, 6) = "x"
pond(5, 6) = "x"
```

^^Testing with 3 positions.

```
pond(5, 5) = "x"
pond(6, 6) = "x"
pond(5, 6) = "x"
pond(1, 1) = "x"
```

^^Testing count algorithms in corners.

```
For counterx = 1 To 10
  For country = 1 To 10
    outputline = pond(counterx, country)
  Next
  Console.WriteLine(outputline)
  outputline = ""
Next
```

^^Error index generation

```
For counterx = 1 To 10
  For country = 1 To 10
    outputline = outputline &
      pond(counterx, country)
  Next
  Console.WriteLine(outputline)
  outputline = ""
Next
```

^^Solved, error was I hadn't accounted for 2 neighbours

```

For counterx = 1 To 10
  For country = 1 To 10
    outputline = outputline &
      pond(counterx, country)
  Next
  Console.WriteLine(outputline)
  outputline = ""
Next

```

^^2 generation showing growth

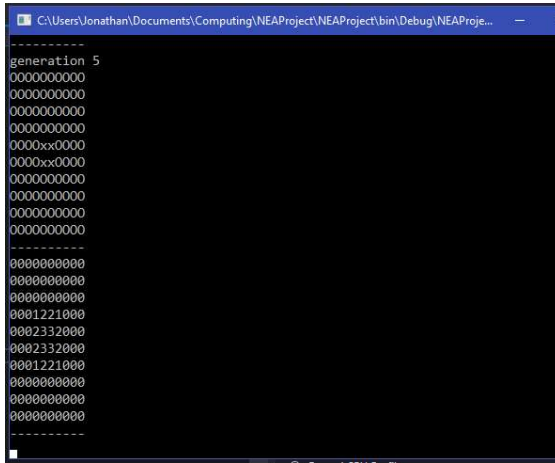
```

Console.WriteLine("-----")
  For counterx = 1 To 10
    For country = 1 To 10
      pond(counterx, country) = nextgen(counterx,
country)
      neighbours(counterx, country) = 0
    Next
  Next
  generation = generation + 1
  System.Threading.Thread.Sleep(1000)
Next gen
Console.ReadLine()
End Sub

```

^^Testing generation 1

^^First attempt at 5 generations with a few bugs



```
-----  
generation 5  
00000000  
00000000  
00000000  
00000000  
00000000  
0000xx0000  
0000xx0000  
00000000  
00000000  
00000000  
00000000  
-----  
00000000  
00000000  
00000000  
0001221000  
0002332000  
0001221000  
00000000  
00000000  
00000000  
-----
```

^^Bug fixed neighbour count needed reset after each iteration

First working prototype:

```

Module Module1
Sub Main()
'initialising variables
Dim outputline As String
Dim pond(11, 11) As String
Dim neighbours(10, 10) As Integer
Dim nextgen(10, 10) As String
Dim generation As Integer = 1
For counterx = 1 To 10
    For country = 1 To 10
        pond(counterx, country) = "0"
        neighbours(counterx, country) = 0
    Next
Next
pond(5, 5) = "x"
pond(6, 6) = "x"
pond(5, 6) = "x"
pond(1, 1) = "x"
For gen = 1 To 20
    Console.Clear()
    For counterx = 1 To 10
        For country = 1 To 10
            If pond((counterx - 1), (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx - 1), country) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx - 1), (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond(counterx, (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond(counterx, (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), country) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
        Next
    Next
    'outputroutine
    Console.WriteLine("Generation " & generation)
    outputline = ""
    For counterx = 1 To 10
        For country = 1 To 10
            outputline = outputline & pond(counterx, country)
        Next
        Console.WriteLine(outputline)
        outputline = ""
    Next
    Console.WriteLine("-----")
    outputline = ""
    'For counterx = 1 To 10
    '    For country = 1 To 10
    '        outputline = outputline & neighbours(counterx, country)
    '    Next
    '    Console.WriteLine(outputline)
    '    outputline = ""
    'Next

```

```
For counterx = 1 To 10
  For country = 1 To 10
    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
      nextgen(counterx, country) = "0"
    End If
    If neighbours(counterx, country) = 3 Then
      nextgen(counterx, country) = "x"
    End If
    If neighbours(counterx, country) = 2 Then
      If pond(counterx, country) = "x" Then
        nextgen(counterx, country) = "x"
      Else
        nextgen(counterx, country) = "0"
      End If
    End If
  Next
Next
Console.WriteLine("-----")
' outputline = ""
'For counterx = 1 To 10
'For country = 1 To 10
'outputline = outputline & nextgen(counterx, country)
'Next
'Console.WriteLine(outputline)
'outputline = ""
'Next
For counterx = 1 To 10
  For country = 1 To 10
    pond(counterx, country) = nextgen(counterx, country)
    neighbours(counterx, country) = 0
  Next
Next
generation = generation + 1
System.Threading.Thread.Sleep(1000)
Next gen
Console.ReadLine()
End Sub
End Module
```

^^Full code in console

Changing the Visual Speed Output

```

'initialising variables
Dim outputline As String
Dim pond(11, 11) As String
Dim neighbours(10, 10) As Integer
Dim nextgen(10, 10) As String
Dim generation As Integer = 1
For counterx = 1 To 10
    For country = 1 To 10
        pond(counterx, country) = "0"
        neighbours(counterx, country) = 0
    Next
Next
pond(5, 5) = "x"
pond(6, 6) = "x"
pond(5, 6) = "x"
pond(1, 1) = "x"
For gen = 1 To 20
    Console.Clear()
    For counterx = 1 To 10
        For country = 1 To 10
            If pond((counterx - 1), (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx,
country) + 1
            End If
            If pond((counterx - 1), country) = "x" Then
                neighbours(counterx, country) = neighbours(counterx,
country) + 1
            End If
            If pond((counterx - 1), (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx,
country) + 1
            End If
            If pond(counterx, (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx,
country) + 1
            End If
            If pond(counterx, (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx,
country) + 1
            End If
        Next
    Next
    Console.WriteLine("Generation " & gen)
    For counterx = 1 To 10
        For country = 1 To 10
            Console.Write(pond(counterx, country) & " ")
        Next
        Console.WriteLine()
    Next
    Console.WriteLine("-----")
    For counterx = 1 To 10
        For country = 1 To 10
            Console.Write(neighbours(counterx, country) & " ")
        Next
        Console.WriteLine()
    Next
    Console.WriteLine("-----")
    For counterx = 1 To 10
        For country = 1 To 10
            Console.Write(nextgen(counterx, country) & " ")
        Next
        Console.WriteLine()
    Next
    generation = gen + 1
Next

```

^^console.clear added to the loop to create a illusion of animation but too fast for the human eye

```

'outputroutine
    Console.WriteLine("Generation " & generation)
    outputline = ""
    For counterx = 1 To 10
        For country = 1 To 10
            outputline = outputline & pond(counterx, country)
        Next
        Console.WriteLine(outputline)
        outputline = ""
    Next
    Console.WriteLine("-----")
    outputline = ""

    For counterx = 1 To 10
        For country = 1 To 10
            If neighbours(counterx, country) < 2 Or neighbours(counterx,
country) > 3 Then
                nextgen(counterx, country) = "0"
            End If
            If neighbours(counterx, country) = 3 Then
                nextgen(counterx, country) = "x"
            End If
            If neighbours(counterx, country) = 2 Then
                If pond(counterx, country) = "x" Then
                    nextgen(counterx, country) = "x"
                Else
                    nextgen(counterx, country) = "0"
                End If
            End If
        Next
    Next
    Console.WriteLine("-----")
    For counterx = 1 To 10
        For country = 1 To 10
            pond(counterx, country) = nextgen(counterx, country)
            neighbours(counterx, country) = 0
        Next
    Next
    generation = generation + 1
    System.Threading.Thread.Sleep(1000)
Next gen
Console.ReadLine()
End Sub

```

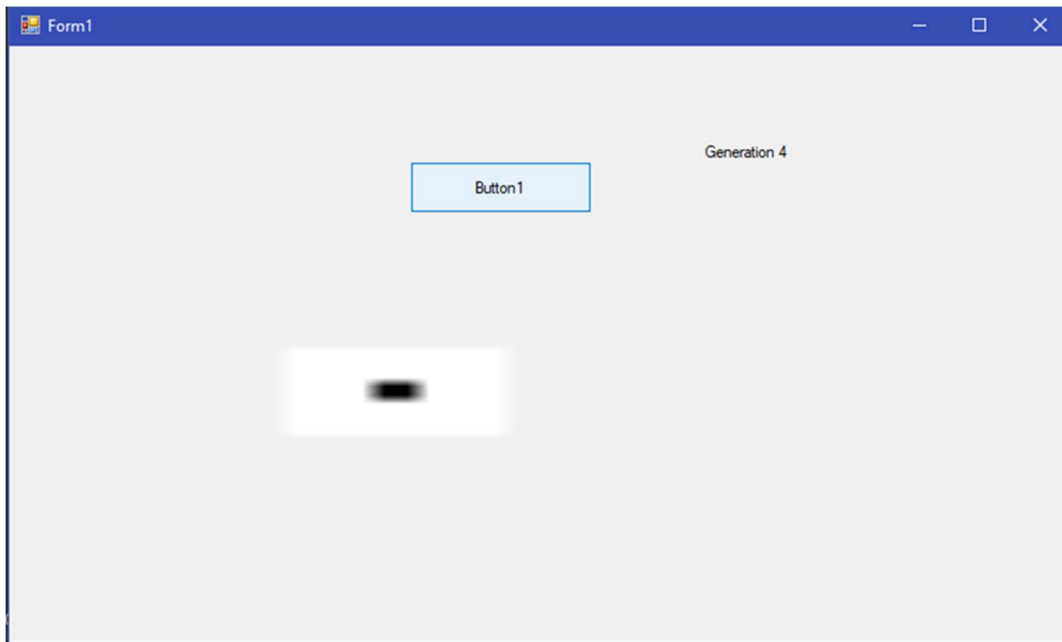
^^Solved, system.threading.thread.sleep(1000) creates the illusion of animation but slow enough to be seen by human eye

Moving the project to Forms:

Once I had a working solution using the console, I started to port the solution to VB forms. This was quite difficult as I had never used forms before, so the first this I had to do was to upskill myself.

Once I had a good understanding of the 'Forms' environment I realised that there were two main routes that I could go down in order to create and animated display. I could either use drawing tools, or manipulate an image at the pixel level. I decided to try some prototyping with the pixel method as I thought this would have a sharper's resolution, support larger population sizes and facilitate resizing, zooming into detail etc.

However, my initial attempt was disappointing.



The picture below shows a 6x6 matrix with each element in the matrix represented by one pixel as can be seen by the diagram visual studio does not scale pixels well.

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        Dim a, c, q, p, r, s As Integer
        Dim picout As New Bitmap(20, 20,
System.Drawing.Imaging.PixelFormat.Format32bppPArgb)
        Dim cw As New Color
        cw = Color.Black
        For y = 0 To 6
            For x = 0 To 6
                picout.SetPixel(y, x, cw)
                If cw = Color.Black Then
                    cw = Color.White
                Else
                    cw = Color.Black
                End If
            Next
        Next
        PictureBox1.Image = picout
        Me.Refresh()
        cw = Color.White
    End Sub
End Class
```



^^After some thought decided to try using more pixels per bacterium. This gave a much sharper output.



This seemd to be a workable solution, so I went on to develop an intermediate solution with it using hard coded initial conditions for now

```

Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim pond(11, 11) As String
        Dim neighbours(10, 10) As Integer
        Dim nextgen(10, 10) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        'new
        Dim outputpic As New Bitmap(200, 200, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
        Dim clr As New Color
        For counterx = 1 To 10
            For country = 1 To 10
                pond(counterx, country) = "0"
                neighbours(counterx, country) = 0
            Next
        Next
        pond(5, 5) = "x"
        pond(6, 6) = "x"
        pond(5, 6) = "x"
        pond(4, 5) = "x"
        For gen = 1 To 20
            'calculates next generation
            For counterx = 1 To 10
                For country = 1 To 10
                    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
                        nextgen(counterx, country) = "0"
                        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                            For subcountry = (10 * country) To (10 * country + blocksize)
                                outputpic.SetPixel(subcounterx, subcountry, Color.White)
                            Next
                        Next
                    End If
                    If neighbours(counterx, country) = 3 Then
                        nextgen(counterx, country) = "x"
                        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                            For subcountry = (10 * country) To (10 * country + blocksize)
                                outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                            Next
                        Next
                    End If
                    If neighbours(counterx, country) = 2 Then
                        If pond(counterx, country) = "x" Then
                            nextgen(counterx, country) = "x"
                            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                                For subcountry = (10 * country) To (10 * country + blocksize)
                                    outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                                Next
                            Next
                        Else
                            nextgen(counterx, country) = "0"
                            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                                For subcountry = (10 * country) To (10 * country + blocksize)
                                    outputpic.SetPixel(subcounterx, subcountry, Color.White)
                                Next
                            Next
                        End If
                    End If
                Next
            Next
        Next
        For counterx = 1 To 10
            For country = 1 To 10
                pond(counterx, country) = nextgen(counterx, country)
                neighbours(counterx, country) = 0
            Next
        Next
        generation = generation + 1
        pctDisplay.Image = outputpic
        Me.Refresh()
        System.Threading.Thread.Sleep(1000)
    Next gen
End Sub
End Class

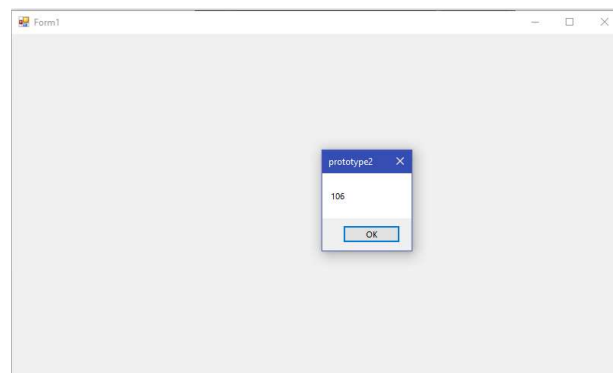
```


Input Method

The next thing I decided to investigate was an input method. The previous code had the initial conditions hard coded into the solution and I wanted to allow the user to set the initial conditions themselves, I found a method using the mouse down event where I could return the x and y coordinates of a picture box.

Prototype

```
Public Class form1
    Private Sub PictureBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PictureBox1.MouseDown
        MsgBox(e.X.ToString)
        MsgBox(e.Y.ToString)
    End Sub
End Class
```



Adding this to my solution

```

Public Class Form1
    Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    Dim pond(51, 51) As String
    Dim maxx As Integer = 50
    Dim maxy As Integer = 50
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim neighbours(maxx, maxy) As Integer
        Dim nextgen(maxx, maxy) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        For x = 1 To maxx
            For y = 1 To maxy
                neighbours(x, y) = 0
            Next
        Next
        Next
        'new
        Dim clr As New Color
        For gen = 1 To 1000
            ' Calculate each cell's number of neighbours
            For counterx = 1 To maxx
                For country = 1 To maxy
                    If pond((counterx - 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                Next
            Next
            lblGeneration.Text = ("Generation " & generation)
            'calculates next generation
            For counterx = 1 To maxx
                For country = 1 To maxy
                    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
                        nextgen(counterx, country) = "0"
                    For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                        For subcountry = (10 * country) To (10 * country + blocksize)
                            outputpic.SetPixel(subcounterx, subcountry, Color.White)
                        Next
                    Next
                End If
            End If
        End For
    End Sub
End Class

```

```

    If neighbours(counterx, country) = 3 Then
        nextgen(counterx, country) = "x"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
            For subcountry = (10 * country) To (10 * country + blocksize)
                outputpic.SetPixel(subcounterx, subcountry, Color.Black)
            Next
        Next
    End If
    If neighbours(counterx, country) = 2 Then
        If pond(counterx, country) = "x" Then
            nextgen(counterx, country) = "x"
            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                For subcountry = (10 * country) To (10 * country + blocksize)
                    outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                Next
            Next
        Else
            nextgen(counterx, country) = "0"
            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                For subcountry = (10 * country) To (10 * country + blocksize)
                    outputpic.SetPixel(subcounterx, subcountry, Color.White)
                Next
            Next
        End If
    End If
Next
counterx = 1 To maxx
country = 1 To maxy
pond(counterx, country) = nextgen(counterx, country)
neighbours(counterx, country) = 0
Next
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
Next gen
End Sub

Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles
pctDisplay.MouseDown
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, country As Integer
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If pond(a, b) = "x" Then
        pond(a, b) = "0"
    Else
        pond(a, b) = "x"
    End
    For x = (a * 10) To (a * 10 + 9)
        For y = (b * 10) To (b * 10 + 9)
            If pond(a, b) = "x" Then
                outputpic.SetPixel(x, y, Color.Black)
            Else
                outputpic.SetPixel(x, y, Color.white)
            End If
        Next
    Next
    pctDisplay.Image = outputpic
End Sub
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    For counterx = 1 To maxx
        For country = 1 To maxy
            pond(counterx, country) = "0"
        Next
    Next

```

```
For k = 1 To 500
  For l = 1 To 500
    outputpic.SetPixel(k, l, Color.White)
  Next
Next

pctDisplay.Image = outputpic
End Sub
End Class
```

Simplifies Output

Showing successful integration of previous console based solutions into forms (simplified output)

```

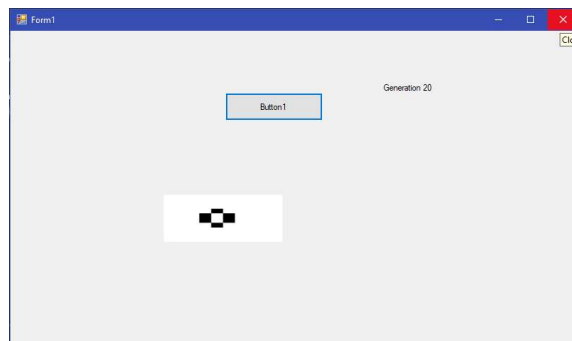
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim pond(11, 11) As String
        Dim neighbours(10, 10) As Integer
        Dim nextgen(10, 10) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        'new
        Dim outputpic As New Bitmap(200, 200,
System.Drawing.Imaging.PixelFormat.Format32bppArgb)
        Dim clr As New Color
        For counterx = 1 To 10
            For country = 1 To 10
                pond(counterx, country) = "0"
                neighbours(counterx, country) = 0
            Next
        Next
        pond(5, 5) = "x"
        pond(6, 6) = "x"
        pond(5, 6) = "x"
        pond(4, 5) = "x"
        For gen = 1 To 20
            For counterx = 1 To 10
                For country = 1 To 10
                    If pond((counterx - 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                Next
            Next
            lblGeneration.Text = ("Generation " & generation)
            outputline = ""
            For counterx = 1 To 10
                For country = 1 To 10
                    outputline = outputline & pond(counterx, country)
                Next
                'MsgBox(outputline)
                outputline = ""
            Next
            outputline = ""
        Next
    End Sub
End Class

```

```

For counterx = 1 To 10
  For country = 1 To 10
    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) >
3 Then
      nextgen(counterx, country) = "0"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.White)
        Next
      Next
    End If
    If neighbours(counterx, country) = 3 Then
      nextgen(counterx, country) = "x"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.Black)
        Next
      Next
    End If
    If neighbours(counterx, country) = 2 Then
      If pond(counterx, country) = "x" Then
        nextgen(counterx, country) = "x"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country +
blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.Black)
          Next
        Next
      Else
        nextgen(counterx, country) = "0"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country +
blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.White)
          Next
        Next
      End If
    End If
  Next
Next
For counterx = 1 To 10
  For country = 1 To 10
    pond(counterx, country) = nextgen(counterx, country)
    neighbours(counterx, country) = 0
  Next
Next
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
System.Threading.Thread.Sleep(1000)
Next gen
End Sub
End Class

```



Integrating the Input Routine

```

Public Class Form1

    Dim pond(11, 11) As String
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outline As String

        Dim neighbours(10, 10) As Integer
        Dim nextgen(10, 10) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        'new
        Dim outputpic As New Bitmap(200, 200,
System.Drawing.Imaging.PixelFormat.Format32bppArgb)
        Dim clr As New Color
        For counterx = 1 To 10
            For country = 1 To 10
                pond(counterx, country) = "0"
                neighbours(counterx, country) = 0
            Next
        Next
        pond(5, 5) = "x"
        pond(6, 6) = "x"
        pond(5, 6) = "x"
        pond(4, 5) = "x"
        pond(4, 4) = "x"
        For gen = 1 To 20
            ' Calculate each cell's number of neighbours
            For counterx = 1 To 10
                For country = 1 To 10
                    If pond((counterx - 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                Next
            Next
            lblGeneration.Text = ("Generation " & generation)
            outline = ""
            For counterx = 1 To 10
                For country = 1 To 10
                    outline = outline & pond(counterx, country)
                Next
            Next
            'MsgBox(outline)
            outline = ""
        Next
    End Sub
End Class

```

```

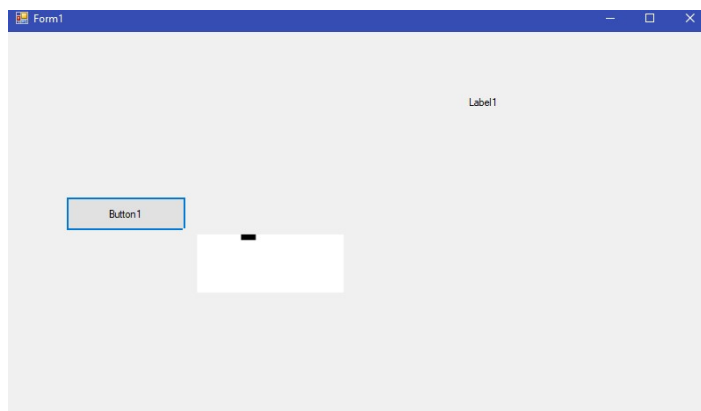
'calculates next generation
For counterx = 1 To 10
  For country = 1 To 10
    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
      nextgen(counterx, country) = "0"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.White)
        Next
      Next
    End If
    If neighbours(counterx, country) = 3 Then
      nextgen(counterx, country) = "x"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.Black)
        Next
      Next
    End If
    If neighbours(counterx, country) = 2 Then
      If pond(counterx, country) = "x" Then
        nextgen(counterx, country) = "x"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country + blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.Black)
          Next
        Next
      Else
        nextgen(counterx, country) = "0"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country + blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.White)
          Next
        Next
      End If
    End If
  Next
Next
For counterx = 1 To 10
  For country = 1 To 10
    pond(counterx, country) = nextgen(counterx, country)
    neighbours(counterx, country) = 0
  Next
Next
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
System.Threading.Thread.Sleep(1000)
Next gen
End Sub

Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles pctDisplay.MouseDown
  Dim a As Integer
  Dim b As Integer
  Dim blocksize As Integer = 9
  Dim counterx, country As Integer
  Dim outputpic As New Bitmap(200, 200, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
  MsgBox(e.X.ToString)
  MsgBox(e.Y.ToString)
  a = (e.X.ToString)
  b = (e.Y.ToString)
  a = Int(a / 10)
  b = Int(b / 10)
  pond(a, b) = "x"

  For x = 1 To 10
    For y = 1 To 10
      For h = (x * 10) To ((x * 10) + 9)
        For j = (y * 10) To ((y * 10) + 9)
          If pond(x, y) = "x" Then
            outputpic.SetPixel(h, j, Color.Black)
          Else
            outputpic.SetPixel(h, j, Color.White)
          End If
        Next
      Next
    Next
  Next
  pctDisplay.Image = outputpic

  'PictureBox1.Image = outputpic
End Sub
End Class

```



Finished Base Code

```

Public Class Form1
    Dim pond(101, 101) As String
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim neighbours(100, 100) As Integer
        Dim nextgen(100, 100) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        'new
        Dim outputpic As New Bitmap(1100, 1100, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
        Dim clr As New Color
        Dim gen = 1 To 1000
        ' Calculate each cell's number of neighbours
        For counterx = 1 To 100
            For country = 1 To 100
                If pond((counterx - 1), (country - 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond((counterx - 1), country) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond((counterx - 1), (country + 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond(counterx, (country - 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond(counterx, (country + 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond((counterx + 1), (country - 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond((counterx + 1), country) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
                If pond((counterx + 1), (country + 1)) = "x" Then
                    neighbours(counterx, country) = neighbours(counterx, country) + 1
                End If
            Next
        Next
        lblGeneration.Text = ("Generation " & generation)
        outputline = ""
        For counterx = 1 To 100
            For country = 1 To 100
                outputline = outputline & pond(counterx, country)
            Next
            'MsgBox(outputline)
            outputline = ""
        Next
        outputline = ""
        'calculates next generation
        For counterx = 1 To 100
            For country = 1 To 100
                If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
                    nextgen(counterx, country) = "0"
                    For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                        For subcountry = (10 * country) To (10 * country + blocksize)
                            outputpic.SetPixel(subcounterx, subcountry, Color.White)
                        Next
                    Next
                End If
                If neighbours(counterx, country) = 3 Then
                    nextgen(counterx, country) = "x"
                    For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                        For subcountry = (10 * country) To (10 * country + blocksize)
                            outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                        Next
                    Next
                End If
            Next
        Next
    End Sub
End Class

```

```

If neighbours(counterx, country) = 2 Then
    If pond(counterx, country) = "x" Then
        nextgen(counterx, country) = "x"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
            For subcountry = (10 * country) To (10 * country + blocksize)
                outputpic.SetPixel(subcounterx, subcountry, Color.Black)
            Next
        Next
    Else
        nextgen(counterx, country) = "0"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
            For subcountry = (10 * country) To (10 * country + blocksize)
                outputpic.SetPixel(subcounterx, subcountry, Color.White)
            Next
        Next
    End If
Next
End Sub

Next
For counterx = 1 To 100
    For country = 1 To 100
        pond(counterx, country) = nextgen(counterx, country)
        neighbours(counterx, country) = 0
    Next
Next
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
System.Threading.Thread.Sleep(1000)
Next gen
End Sub

Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles pctDisplay.MouseDown
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, country As Integer
    Dim outputpic As New Bitmap(1100, 1100, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    'MsgBox(e.X.ToString)
    'MsgBox(e.Y.ToString)
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If pond(a, b) = "x" Then
        pond(a, b) = "o"
    Else
        pond(a, b) = "x"
    End If

    For x = 0 To 99
        For y = 0 To 99
            For h = (x * 10) To ((x * 10) + 9)
                For j = (y * 10) To ((y * 10) + 9)
                    If pond(x, y) = "x" Then
                        outputpic.SetPixel(h, j, Color.Black)
                    Else
                        outputpic.SetPixel(h, j, Color.White)
                    End If
                Next
            Next
        Next
    Next

    pctDisplay.Image = outputpic

    'PictureBox1.Image = outputpic
End Sub

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    For counterx = 1 To 100
        For country = 1 To 100
            pond(counterx, country) = "0"
            neighbours(counterx, country) = 0
        Next
    Next
    pond(5, 5) = "x"
    pond(6, 6) = "x"
    pond(5, 6) = "x"
    pond(4, 5) = "x"
    pond(4, 4) = "x"
End Sub
End Class
Trying to increase board size

```

```

Public Class Form1
    Dim pond(51, 51) As String
    Dim maxx As Integer = 50
    Dim maxy As Integer = 50
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim neighbours(maxx, maxy) As Integer
        Dim nextgen(maxx, maxy) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9
        'new
        Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
        Dim clr As New Color
        For gen = 1 To 1000
            ' Calculate each cell's number of neighbours
            For counterx = 1 To maxx
                For country = 1 To maxy
                    If pond((counterx - 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx - 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond(counterx, (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country - 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), country) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                    If pond((counterx + 1), (country + 1)) = "x" Then
                        neighbours(counterx, country) = neighbours(counterx, country) + 1
                    End If
                Next
            Next
            lblGeneration.Text = ("Generation " & generation)
            'calculates
            next generation
            For counterx = 1 To maxx
                For country = 1 To maxy
                    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
                        nextgen(counterx, country) = "0"
                        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                            For subcountry = (10 * country) To (10 * country + blocksize)
                                outputpic.SetPixel(subcounterx, subcountry, Color.White)
                            Next
                        Next
                    End If
                    If neighbours(counterx, country) = 3 Then
                        nextgen(counterx, country) = "x"
                        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                            For subcountry = (10 * country) To (10 * country + blocksize)
                                outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                            Next
                        Next
                    End If
                    If neighbours(counterx, country) = 2 Then
                        If pond(counterx, country) = "x" Then
                            nextgen(counterx, country) = "x"
                            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                                For subcountry = (10 * country) To (10 * country + blocksize)
                                    outputpic.SetPixel(subcounterx, subcountry, Color.Black)
                                Next
                            Next
                        Else
                            nextgen(counterx, country) = "0"
                            For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
                                For subcountry = (10 * country) To (10 * country + blocksize)
                                    outputpic.SetPixel(subcounterx, subcountry, Color.White)
                                Next
                            Next
                        End If
                    End If
                Next
            Next
            For counterx = 1 To maxx
                For country = 1 To maxy
                    pond(counterx, country) = nextgen(counterx, country)
                    neighbours(counterx, country) = 0
                Next
            Next
            generation = generation + 1
            pctDisplay.Image = outputpic
            Me.Refresh()
            System.Threading.Thread.Sleep(1000)
        Next
    End Sub

```

```

Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles pctDisplay.MouseDown
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, countery As Integer
    Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    'MsgBox(e.X.ToString)
    'MsgBox(e.Y.ToString)
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If pond(a, b) = "x" Then
        pond(a, b) = "o"
    Else
        pond(a, b) = "x"
    End If

    For x = 0 To maxx
        For y = 0 To maxy
            For h = (x * 10) To ((x * 10) + 9)
                For j = (y * 10) To ((y * 10) + 9)
                    If pond(x, y) = "x" Then
                        outputpic.SetPixel(h, j, Color.Black)
                    Else
                        outputpic.SetPixel(h, j, Color.White)
                    End If
                Next
            Next
        Next
    Next

    pctDisplay.Image = outputpic

    'PictureBox1.Image = outputpic
End Sub

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    For counterx = 1 To maxx
        For countery = 1 To maxy
            pond(counterx, countery) = "0"
            ' neighbours(counterx, countery) = 0
        Next
    Next
    'pond(5, 5) = "x"
    'pond(6, 6) = "x"
    'pond(5, 6) = "x"
    'pond(4, 5) = "x"
    'pond(4, 4) = "x"
End Sub
End Class
So now the board size is a variable

**

Public Class Form1
    Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    Dim pond(51, 51) As String
    Dim maxx As Integer = 50
    Dim maxy As Integer = 50
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim outputline As String
        Dim neighbours(maxx, maxy) As Integer
        Dim nextgen(maxx, maxy) As String
        Dim generation As Integer = 1
        Dim blocksize As Integer = 9

        'outputline = ""
        'For x = 1 To 50
        '    For y = 1 To 50
        '        outputline = outputline & pond(x, y)
        '    Next
        'Next
        'MsgBox(outputline)

        For x = 1 To maxx
            For y = 1 To maxy
                neighbours(x, y) = 0
            Next
        Next
    End Sub
End Class

```

```

'new
' Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
Dim clr As New Color
For gen = 1 To 1000
' Calculate each cell's number of neighbours
For counterx = 1 To maxx
For country = 1 To maxy
If pond((counterx - 1), (country - 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond((counterx - 1), country) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond((counterx - 1), (country + 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond(counterx, (country - 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond(counterx, (country + 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond((counterx + 1), (country - 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond((counterx + 1), country) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
If pond((counterx + 1), (country + 1)) = "x" Then
neighbours(counterx, country) = neighbours(counterx, country) + 1
End If
Next
Next
lblGeneration.Text = ("Generation " & generation)

'calculates next generation
For counterx = 1 To maxx
For country = 1 To maxy
If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
nextgen(counterx, country) = "0"
For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
For subcountry = (10 * country) To (10 * country + blocksize)
outputpic.SetPixel(subcounterx, subcountry, Color.White)
Next
Next
End If
If neighbours(counterx, country) = 3 Then
nextgen(counterx, country) = "x"
For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
For subcountry = (10 * country) To (10 * country + blocksize)
outputpic.SetPixel(subcounterx, subcountry, Color.Black)
Next
Next
End If
If neighbours(counterx, country) = 2 Then
If pond(counterx, country) = "x" Then
nextgen(counterx, country) = "x"
For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
For subcountry = (10 * country) To (10 * country + blocksize)
outputpic.SetPixel(subcounterx, subcountry, Color.Black)
Next
Next
Else
nextgen(counterx, country) = "0"
For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
For subcountry = (10 * country) To (10 * country + blocksize)
outputpic.SetPixel(subcounterx, subcountry, Color.White)
Next
Next
End If
End If
Next
Next
For counterx = 1 To maxx
For country = 1 To maxy
pond(counterx, country) = nextgen(counterx, country)
neighbours(counterx, country) = 0
Next
Next
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
'System.Threading.Thread.Sleep(1000)

```

```

Next gen
End Sub

Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles pctDisplay.MouseDown
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, country As Integer
    ' Dim outputpic As New Bitmap(520, 520, System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    'MsgBox(e.X.ToString)
    'MsgBox(e.Y.ToString)
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If pond(a, b) = "x" Then
        pond(a, b) = "0"
    Else
        pond(a, b) = "x"
    End If
    'MsgBox(a)
    'MsgBox(b)
    'MsgBox(pond(a, b))

    For x = (a * 10) To (a * 10 + 9)
        For y = (b * 10) To (b * 10 + 9)
            If pond(a, b) = "x" Then
                outputpic.SetPixel(x, y, Color.Black)
            Else
                outputpic.SetPixel(x, y, Color.White)
            End If
        Next
    Next

    'For x = 0 To maxx
    '    For y = 0 To maxy
    '        For h = (x * 10) To ((x * 10) + 9)
    '            For j = (y * 10) To ((y * 10) + 9)
    '                If pond(x, y) = "x" Then
    '                    outputpic.SetPixel(h, j, Color.Black)
    '                Else
    '                    outputpic.SetPixel(h, j, Color.White)
    '                End If
    '            Next
    '        Next
    '    Next
    'Next

    pctDisplay.Image = outputpic

    'PictureBox1.Image = outputpic
End Sub

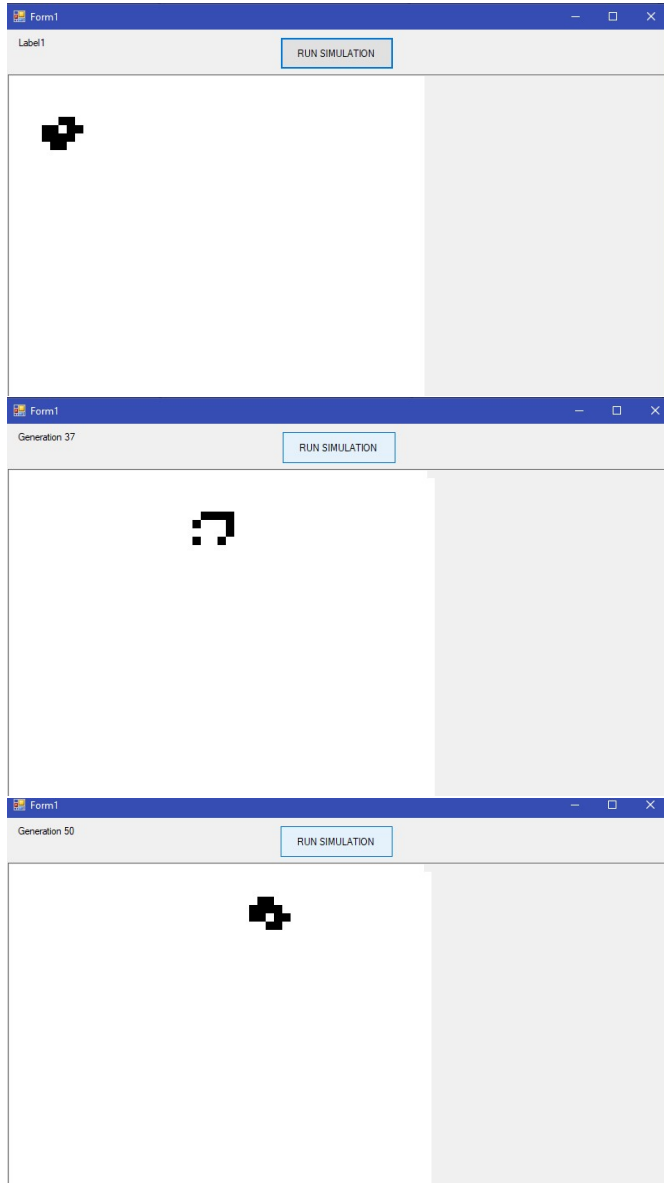
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    For counterx = 1 To maxx
        For country = 1 To maxy
            pond(counterx, country) = "0"
            ' neighbours(counterx, country) = 0
        Next
    Next

    'pond(5, 5) = "x"
    'pond(6, 6) = "x"
    'pond(5, 6) = "x"
    'pond(4, 5) = "x"
    'pond(4, 4) = "x"

    For k = 1 To 500
        For l = 1 To 500
            outputpic.SetPixel(k, l, Color.White)
        Next
    Next

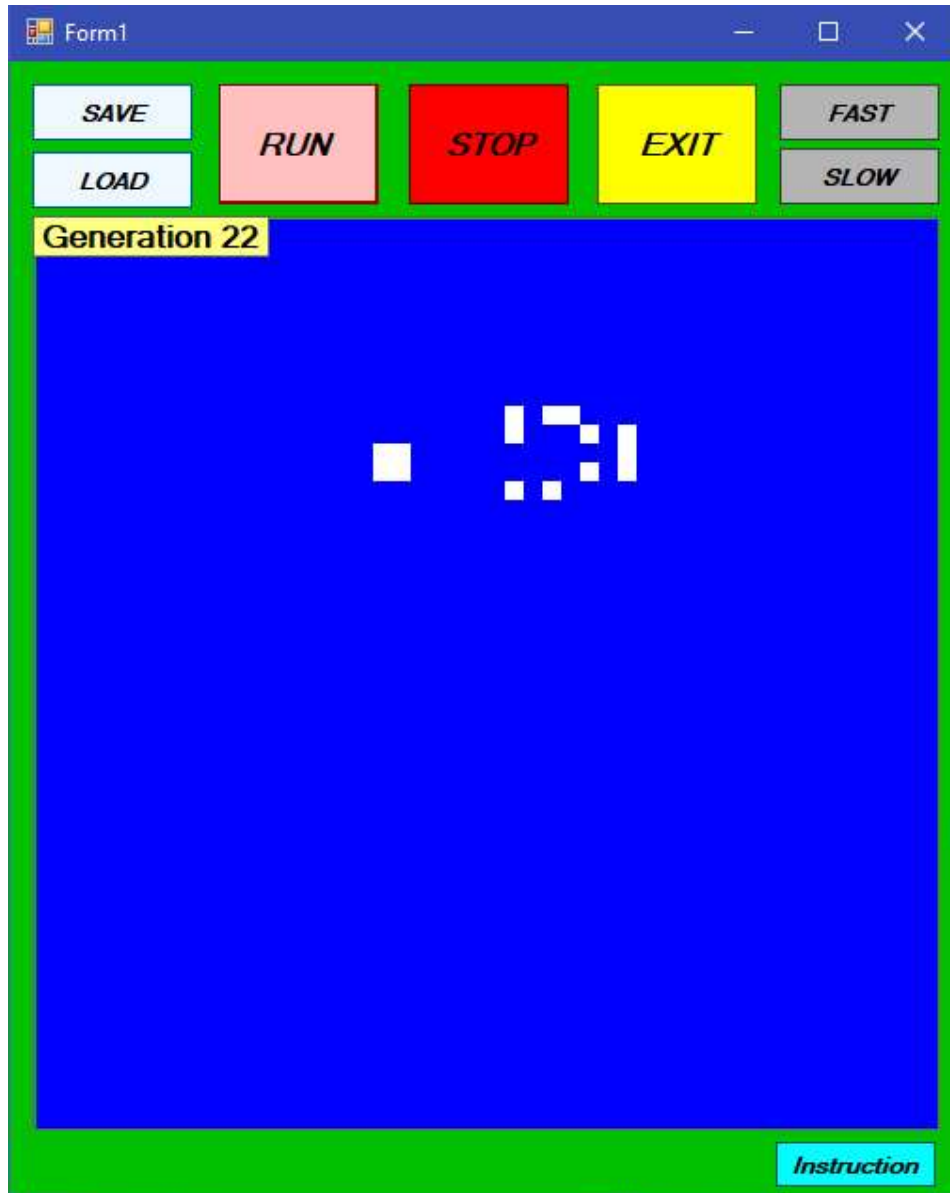
    pctDisplay.Image = outputpic
End Sub
End Class

```

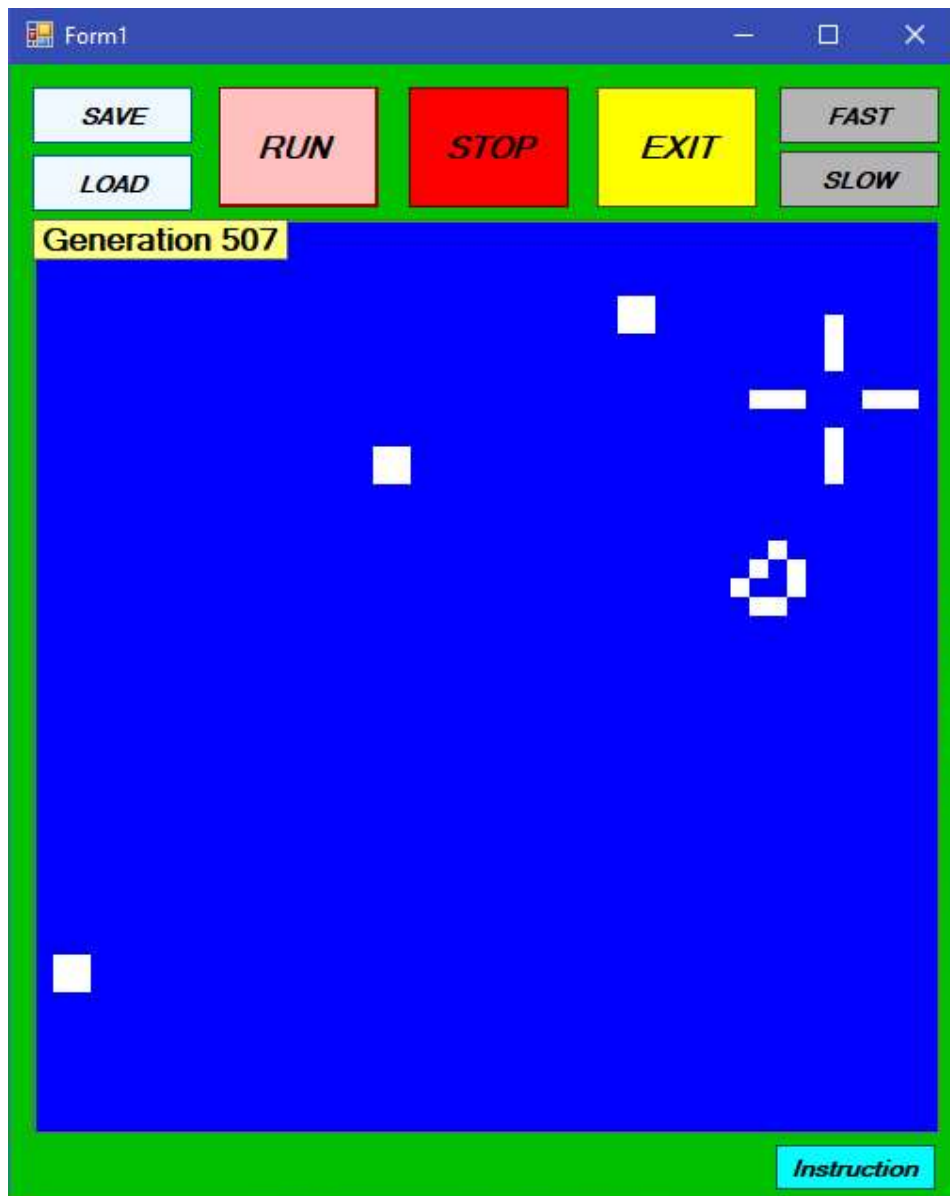


^^ LIFE!! "Life" has been created, this is the program running in action. I went online to look for different designs that can move and found this bird design that moves either to the left or right depending on which way the 'beak' is facing. The response time is perfect, the interactive part of the program works seamlessly, and the regeneration of cells all work with the rules that have been hard coded into the program.

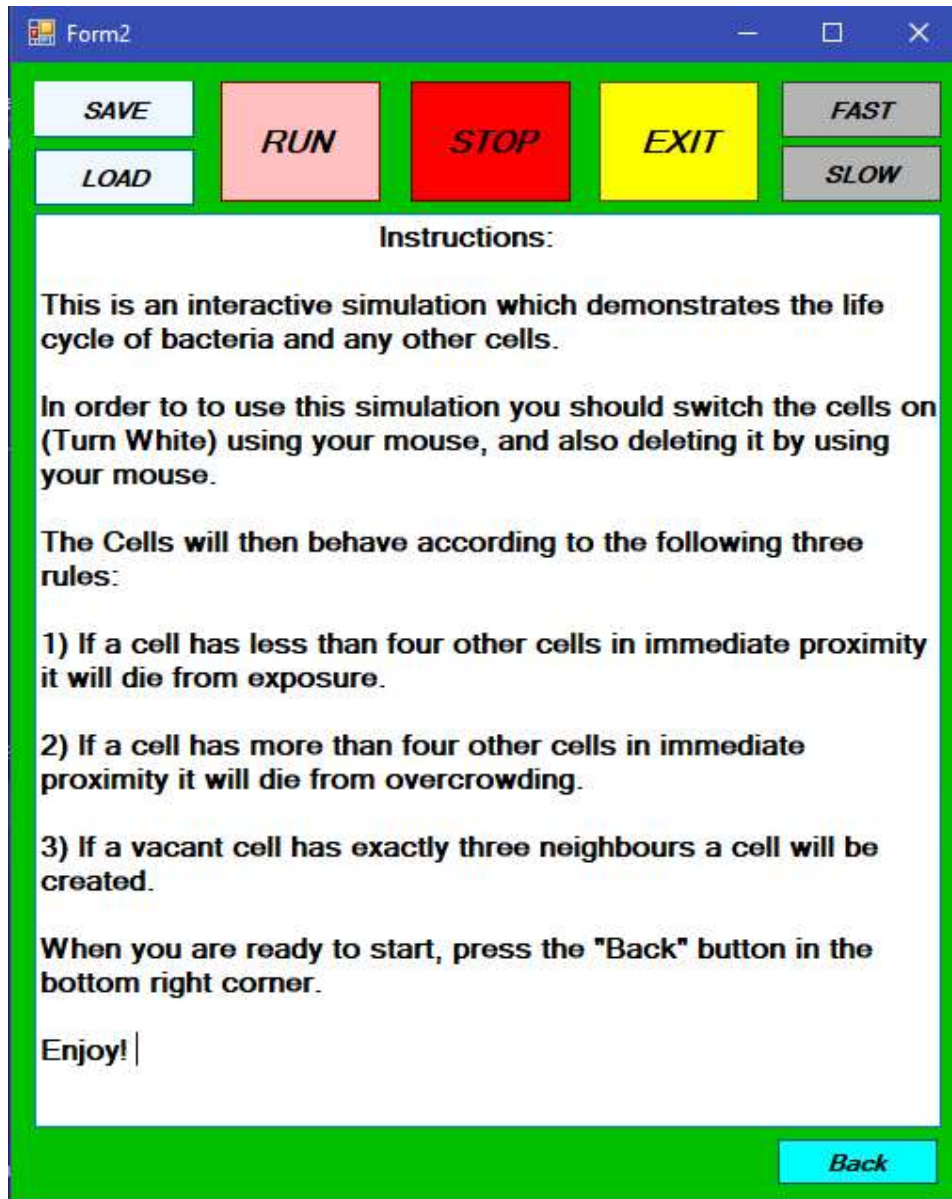
Technical Solution



^^ Finishing touches to the cosmetic appearance of my project. Note the slight difference between solution and design. As I was making the project I realised that it would be easier for me to create a square output rather than a rectangular output, as this kept both my maximum X and Y values equal as I was handling arrays. I do not feel that this design change has detracted from my original design, and even though it would be relatively easy to recode a rectangular solution now I actually prefer the aesthetic of this square.



^^ Testing beyond generation 500 which has proved successful.



^^ Here are my instructions in another form window. Linked the instructions to my main user interface with the following code:

From main user interface to instructions:

```
Private Sub cmdInstructions_Click(sender As Object, e As EventArgs) Handles cmdInstructions.Click
    Form2.Show()
    Me.Hide()
End Sub
```

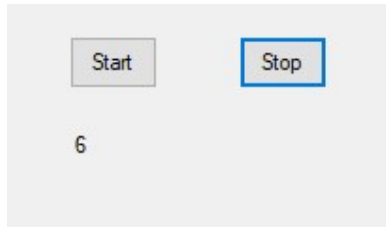
From Instructions back to main user interface:

```
Private Sub cmdback_Click(sender As Object, e As EventArgs) Handles cmdback.Click
    Form1.Show()
    Me.Hide()
End Sub
```

Prototyping the Stop Button by using timer function

In order to add the stop function to my code I needed to take the main processes out of a loop and place them onto the timer function. The timer function will fire at regular intervals and will allow me to attach code to this event. This means that I will be able to check for user activity such as pressing the stop button in between steps. Using my previous method meant that when the start button has been pressed no further user interaction would be detectable.

Prototype



```
Public Class Form1
    Dim count As Integer = 1

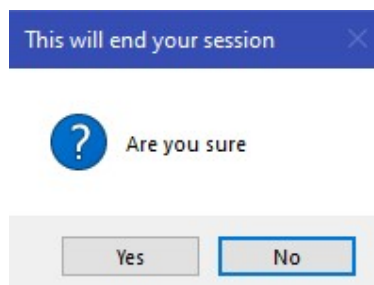
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Timer1.Enabled = True
    End Sub

    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
        Label1.Text = count
        count = count + 1
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        Timer1.Enabled = False
    End Sub
End Class
```

Adding Functionality to Exit Button

I used the inbuilt Visual Basic Yes / No message box in order to check that the user really wanted to terminate the execution.



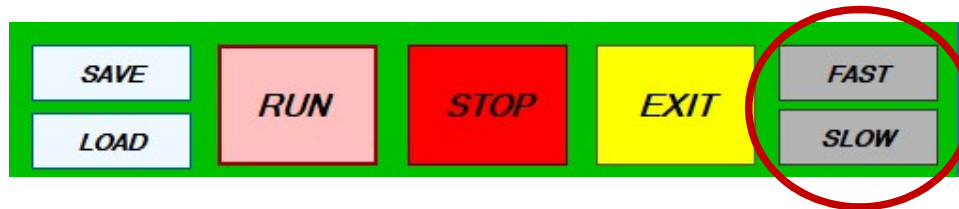
```
Private Sub cmdEXIT_Click(sender As Object, e As EventArgs) Handles cmdEXIT.Click
    Dim answer As Integer
    answer = MsgBox("Are you sure", vbQuestion + vbYesNo + vbDefaultButton2, "This
will end your session")
    If answer = vbYes Then
        Me.Close()
    End If
End Sub
```

Speed Variations

I wrote procedures to allow the user to fine tune the speed of generation progression. This facility is particularly important in order to allow my software to be portable between different machines, as a newer machine will process my code a lot quicker than an older machine and therefore the simulation will appear to go faster or slower depending on (primarily) the class of CPU. I achieved this by linking the speed up and slow down buttons to the timer interval.

```
Private Sub cmdspeedup_Click(sender As Object, e As EventArgs) Handles
cmdspeedup.Click
    Dim timeinterval As Integer
    timeinterval = Tmrtimer.Interval
    timeinterval = timeinterval - 500
    If timeinterval < 500 Then
        timeinterval = 500
    End If
    Tmrtimer.Interval = timeinterval
End Sub
```

```
Private Sub cmdSLOWDOWN_Click(sender As Object, e As EventArgs) Handles
cmdSLOWDOWN.Click
    Dim timeinterval As Integer
    timeinterval = Tmrtimer.Interval
    timeinterval = timeinterval + 500
    If timeinterval > 10000 Then
        timeinterval = 10000
    End If
    Tmrtimer.Interval = timeinterval
End Sub
```



File Handling Prototype

In this section I will demonstrate how I added a "save" and "load" function to my code.

Prototype attempt at using stream writer. I am intending to write three elements to a text file.

```
Imports System.IO
Public Class Form1
    Private Sub save_Click(sender As Object, e As EventArgs)
Handles save.Click
        Dim filewrite As New StreamWriter("Test.txt")
        filewrite.WriteLine("x")
        filewrite.WriteLine("o")
        filewrite.WriteLine("y")
        filewrite.Close()
    End Sub
End Class
```

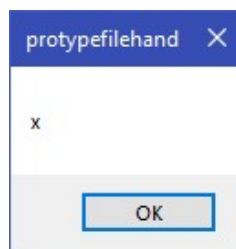


As you can see from the below the text file, the first part of the prototype works.

```
File Edit
x
o
y|
```

Next I tried to read from the text file using stream reader.

```
Dim fileread As StreamReader = New StreamReader("Test.txt")
Dim input1, input2, input3 As String
input1 = fileread.ReadLine
input2 = fileread.ReadLine
input3 = fileread.ReadLine
fileread.Close()
MsgBox(input1)
MsgBox(input2)
MsgBox(input3)
```

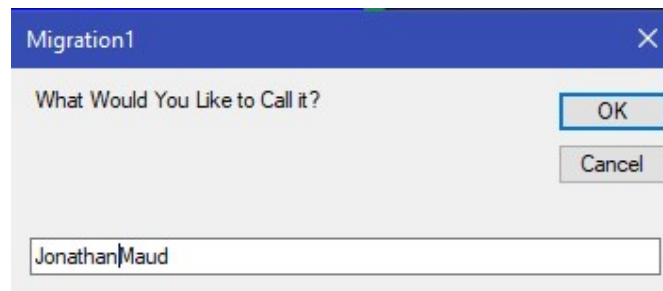


Success!! It works. Next for refinements.

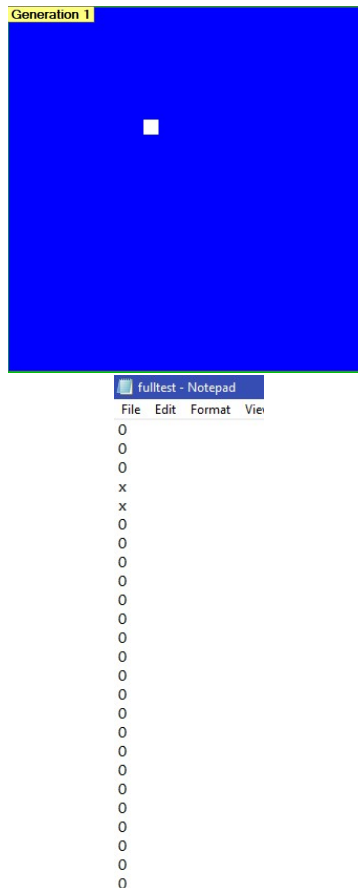
Here I allow the user to choose the name for the saved file. This will mean that the user can save multiple versions.

```
Dim fname As String
fname = InputBox("What Would You Like to Call it?")
fname = fname & ".txt"
Dim filewrite As New StreamWriter(fname)
For xcount = 0 To 50
    For ycount = 0 To 50
        filewrite.WriteLine(pond(xcount, ycount))
    Next
Next
filewrite.Close()
```

Now the user can choose the filename.



Sample output from the text file



Completing the read module to recover the data and update the picture file.

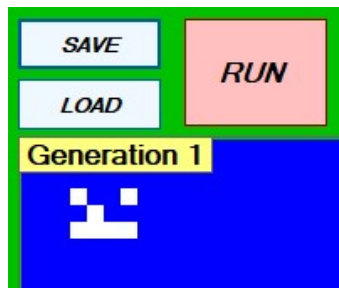
```

Dim a As Integer
Dim b As Integer
Dim blocksize As Integer = 9
Dim counterx, countery As Integer
Dim fname As String
fname = InputBox("Which File Would You Like To Load?")
fname = fname & ".txt"
Dim fileread As StreamReader = New StreamReader(fname)
For xcount = 0 To 50
    For ycount = 0 To 50
        pond(xcount, ycount) = fileread.ReadLine
    Next
Next
fileread.Close()
Dim outputline As String
For xcount = 0 To 50
    For ycount = 0 To 50
        outputline = outputline & pond(xcount, ycount)
    Next
Next
For a = 0 To 50
    For b = 0 To 50
        For x = (a * 10) To (a * 10 + 9)
            For y = (b * 10) To (b * 10 + 9)
                If pond(a, b) = "x" Then
                    outputpic.SetPixel(x, y, Color.White)
                Else
                    outputpic.SetPixel(x, y, Color.Blue)
                End If
            Next
        Next
    Next
Next
pctDisplay.Image = outputpic
End Sub

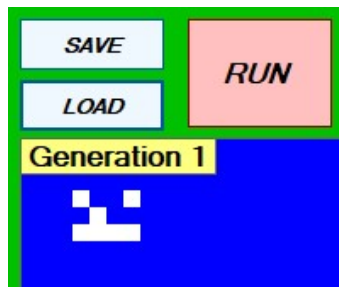
```

Test

Saved version:



Loaded version:



As may be seen both of these screenshots are identical proving that the save and retrieve function is working effectively.

Reset Button

As I reach this point in my design I realised that a reset button will benefit the students and teacher's day to day use.



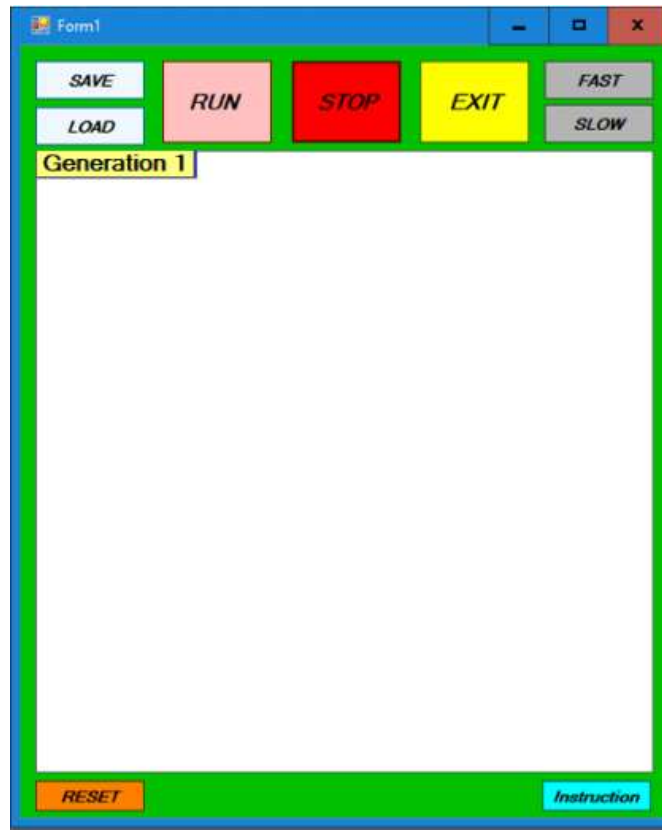
```
Private Sub reset_Click(sender As Object, e As EventArgs) Handles reset.Click
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, country As Integer
    For a = 0 To 50
        For b = 0 To 50
            For x = (a * 10) To (a * 10 + 9)
                For y = (b * 10) To (b * 10 + 9)
                    pond(a, b) = "o"
                    outputpic.SetPixel(x, y, Color.Blue)
                Next
            Next
        Next
    Next
    pctDisplay.Image = outputpic
End Sub
```

As part of the **testing** during the coding process I noticed that when I press the reset button even though the user interface would reset the number of generations would not. I identified the bug and fixed it. Here is my updated code:

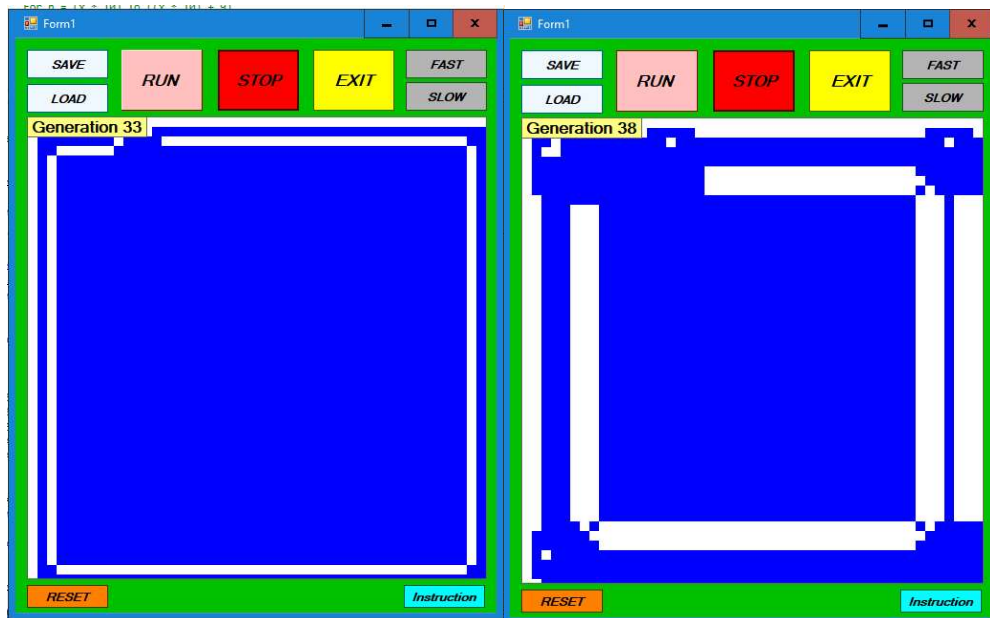
```
Private Sub reset_Click(sender As Object, e As EventArgs) Handles reset.Click
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, country As Integer
    For a = 0 To 50
        For b = 0 To 50
            For x = (a * 10) To (a * 10 + 9)
                For y = (b * 10) To (b * 10 + 9)
                    pond(a, b) = "o"
                    outputpic.SetPixel(x, y, Color.Blue)
                Next
            Next
        Next
    Next
    pctDisplay.Image = outputpic
    generation = 1
    lblGeneration.Text = "Generation " & generation
End Sub
```

Testing for robustness with exceptional data

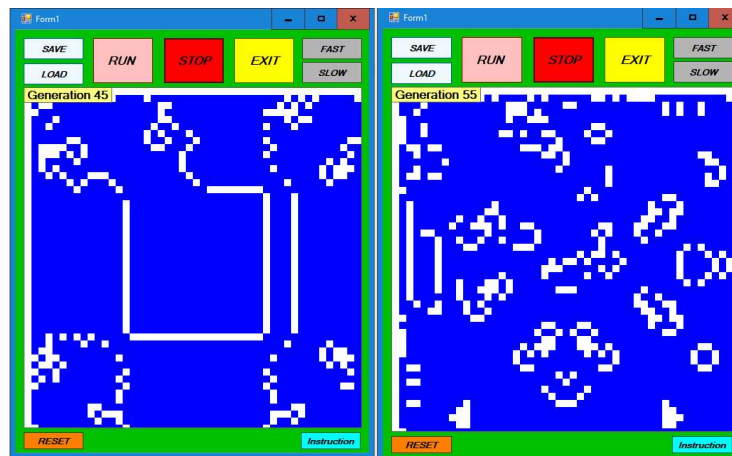
As part of my exceptional testing I decided to fill the whole interface with bacteria to see if my program could handle it.



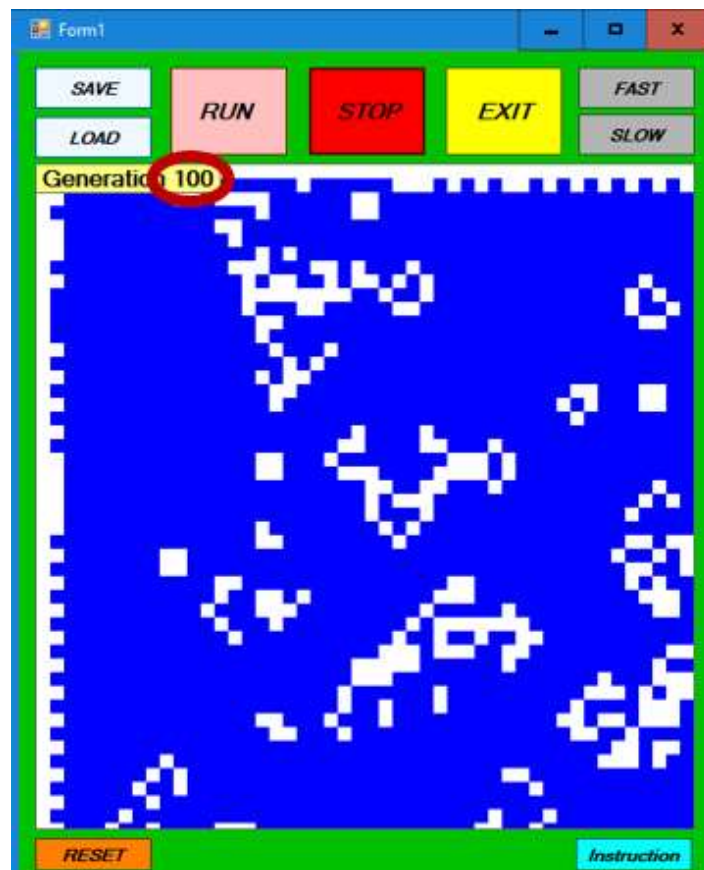
Result – Testing for robustness when coding



As can be seen the test was a success. The 2 screenshots above show the interesting patterns that I received at generation 33 and 38.



The program created new and (in my opinion) beautiful and unexpected animations of the cells providing evidence that my program is successful in simulating life.



Next, I **tested** to see if the program would be able to speed up & slow down using the allocated buttons as well as the stop and run button. Although it's difficult to show the results of these on paper, I tested the results using a stopwatch and they both worked.

Next, I tested to see if the program was able to surpass 100 generations and the result was that it was able to.

The last test I conducted of this was to see if the program reset back to its original state, and so I stopped the program and pressed the reset button and it reset the program without an issue.

Intermediate Stage Feedback

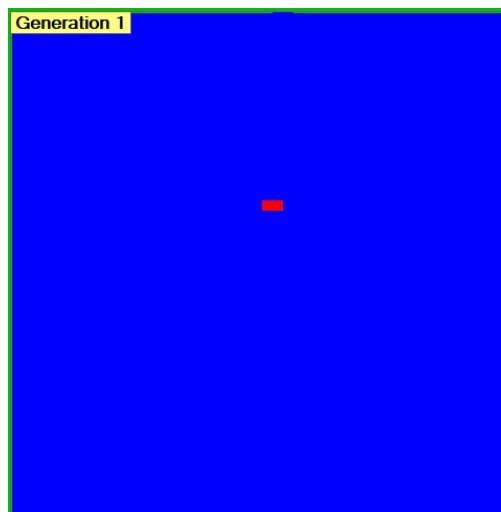
At this stage I met with my end user to take any comments on the program so far. I thought it was close being finished and I wanted to give my end user the opportunity to have further input into the solution. My end user (Mr Watson) was delighted with the program so far and he suggested some further refinements. What Mr Watson requested was to have a series of presets available of standard known patterns in the Game of Life, which would make setting the initial conditions both easier and also would produce more meaningful results than a random input. Mr Watson felt it would be useful to be able to compare and contrast the various forms of "Life" that the program produced with his class, so that he would be able to highlight the salient features of each "Life Form".

Prototyping the Stamp Function

In this section will show how I added preset pattern buttons to my code to try and make my user input more intuitive. My idea was to have a series of buttons down the side of the interface which the user may select. When the user selects the button the tip of the cursor will then deliver a “ghost” replica of the pattern which is being introduced to allow the user to position it correctly. As the user moves the cursor over the display the “ghost” moves with it. Here is my first attempt which is simply 2 square pattern.

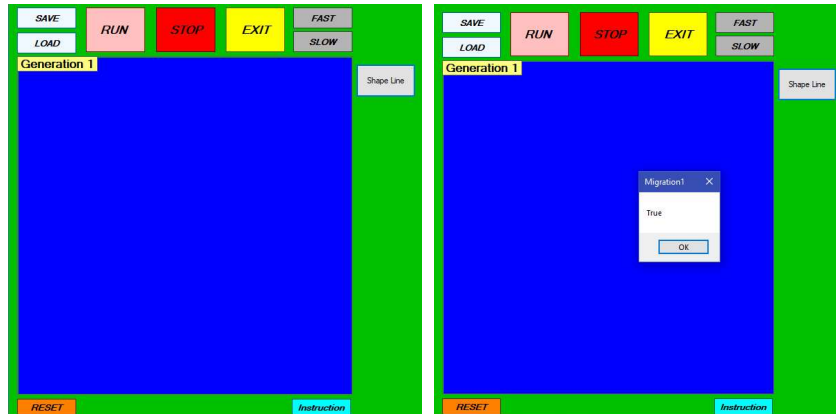
```
Private Sub pctDisplay_MouseMove(sender As Object, e As MouseEventArgs) Handles
pctDisplay.MouseMove
    Dim a, b As Integer
    Dim newa, newb As Integer
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    mousedisplayblue(olda, oldb)
    mousedisplayblue((olda + 1), oldb)
    olda = a
    oldb = b
    mousedisplayred(a, b)
    mousedisplayred((a + 1), b)
    pctDisplay.Image = outputpic
End Sub
Public Sub mousedisplayred(dispa As Integer, dispb As Integer)
    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            outputpic.SetPixel(x, y, Color.Red)
        Next
    Next
End Sub
Public Sub mousedisplayblue(dispa As Integer, dispb As Integer)

    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            outputpic.SetPixel(x, y, Color.Blue)
        Next
    Next
End Sub
```

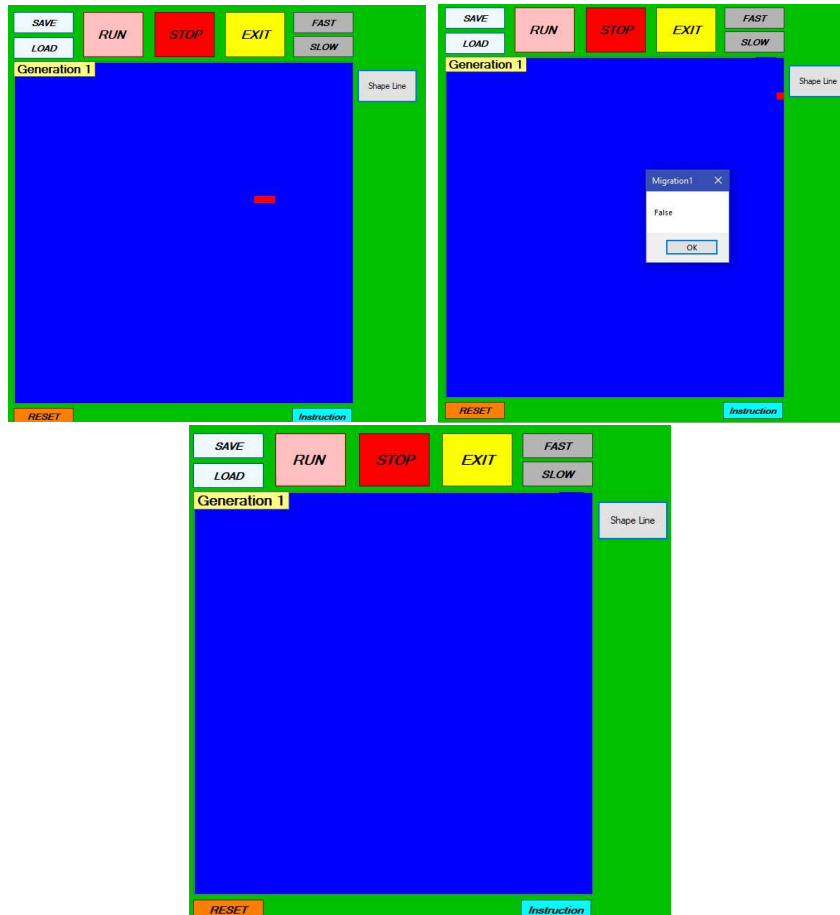


Prototyping the Stamp Tool

Turning the stamp on using the Shape Line Button:



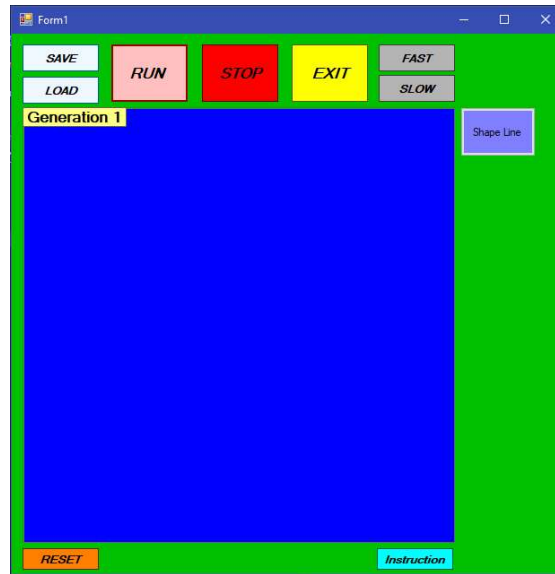
The ghost image is being displayed in red so now turning off the stamp using the same Button:



Partial success however, when the ghost image goes over a "chosen"/ white pixel the white pixel does not reappear as the ghost image passes by.

Presets

Here is my solution to the disappearing pixel problem. I was initially trying handle the issue by using the pixel colours, but eventually I resorted to a refresh from the underlying array (pond) to solve the problem.



```

Private Sub pctCopy_MouseDown(sender As Object, e As MouseEventArgs)
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, countery As Integer
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If lineshape = True Then
        For cntr = 1 To shapecomponent(shapenumber)
            pond((a + xcoordinate(shapenumber, cntr)), (b +
ycoordinate(shapenumber, cntr))) = "x"
        Next
        For cntr = 1 To shapecomponent(shapenumber)
            mousedisplaywhite((olda + xcoordinate(shapenumber, cntr)), (olddb +
ycoordinate(shapenumber, cntr)))
        Next
    Else
        If pond(a, b) = "x" Then
            pond(a, b) = "0"
        Else
            pond(a, b) = "x"
        End If
        For x = (a * 10) To (a * 10 + 9)
            For y = (b * 10) To (b * 10 + 9)
                If pond(a, b) = "x" Then
                    outputpic.SetPixel(x, y, Color.White)
                Else
                    outputpic.SetPixel(x, y, Color.Blue)
                End If
            Next
        Next
    End If
End Sub

```

```
Private Sub pctCopy_MouseMove(sender As Object, e As MouseEventArgs)
    Dim a, b As Integer
    If lineshape = False Then
        Exit Sub
    End If
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    For cntr = 1 To shapecomponent(shapenumber)
        updatedisplay((olda + xcoordinate(shapenumber, cntr)), (oldb +
ycoordinate(shapenumber, cntr)))
    Next
    olda = a
    oldb = b
    For cntr = 1 To shapecomponent(shapenumber)
        mousedisplayred((olda + xcoordinate(shapenumber, cntr)), (oldb +
ycoordinate(shapenumber, cntr)))
    Next
End Sub

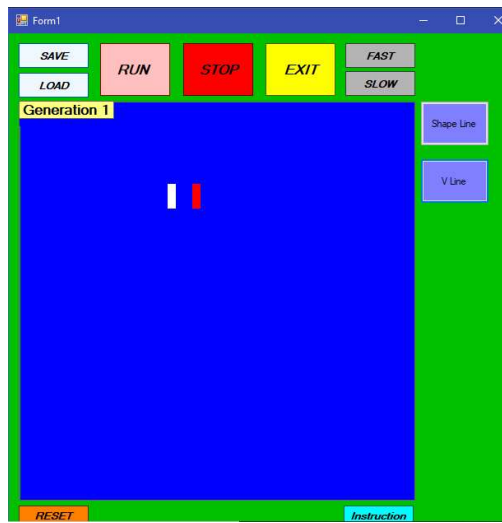
Public Sub updatedisplay(dispa As Integer, dispb As Integer)

    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            If pond(dispa, dispb) = "x" Then
                outputpic.SetPixel(x, y, Color.White)
            Else
                outputpic.SetPixel(x, y, Color.Blue)
            End If
        Next
    Next
End Sub
```

Success! No more disappearing pixels.

Adding New Preset 2

My first preset was a 3 element horizontal line. I decided to make my second a 3 element vertical line.



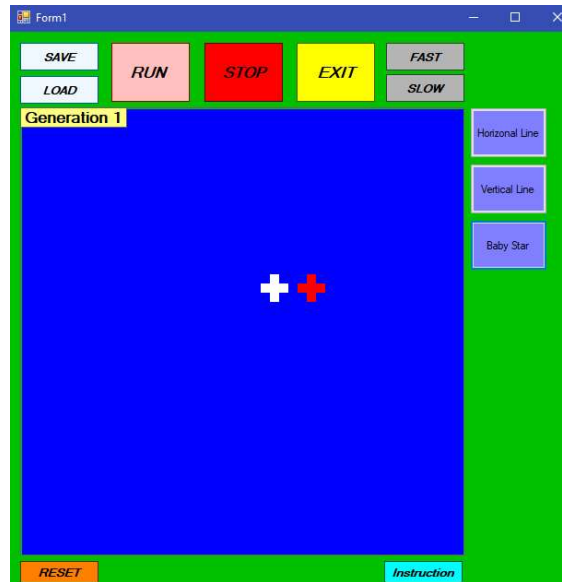
Note the red image is a ghost which follows the cursor tip and helps the user to position the preset stamp. Once the user clicks the selected pixels go white.

```
Private Sub cmdvline_Click(sender As Object, e As EventArgs) Handles
cmdvline.Click
    lineshape = Not (lineshape)
    shapenumber = 2
    shapecomponent(2) = 3
    xcoordinate(2, 1) = 0
    ycoordinate(2, 1) = 0
    xcoordinate(2, 2) = 0
    ycoordinate(2, 2) = 1
    xcoordinate(2, 3) = 0
    ycoordinate(2, 3) = 2
End Sub
```

Having created the first preset meant that the second preset was almost already solved.

Adding New Preset 3

The third preset is called a "Baby Star"



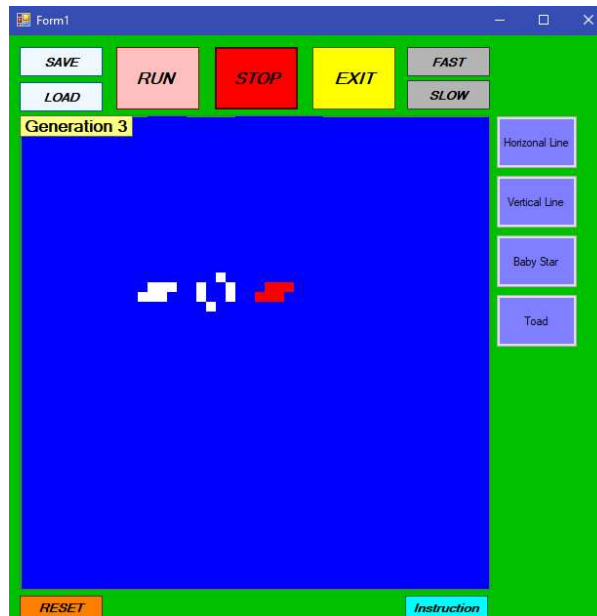
```

Private Sub cmdbs_Click(sender As Object, e As EventArgs) Handles cmdbs.Click
    lineshape = Not (lineshape)
    shapenumber = 3
    shapecomponent(3) = 5
    xcoordinate(3, 1) = 1
    ycoordinate(3, 1) = 0
    xcoordinate(3, 2) = 0
    ycoordinate(3, 2) = 1
    xcoordinate(3, 3) = 1
    ycoordinate(3, 3) = 1
    xcoordinate(3, 4) = 2
    ycoordinate(3, 4) = 1
    xcoordinate(3, 5) = 1
    ycoordinate(3, 5) = 2
End Sub

```

Adding New Preset 4

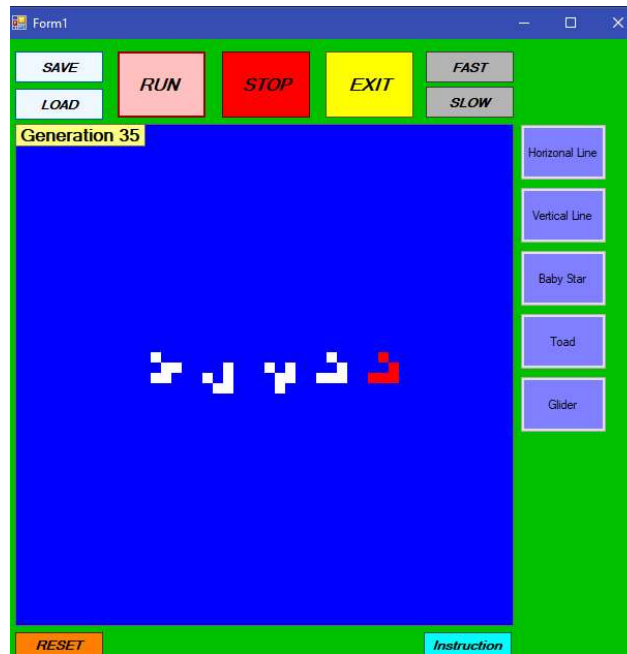
This preset is called an oscillator because it oscillates between two different states, this one is commonly known as a 'Toad'. The screenshot below shows the 2 states that the Toad oscillates between as well as showing the preset ghost image.



```
Private Sub cmdtoad_Click(sender As Object, e As EventArgs) Handles cmdtoad.Click
    lineshape = Not (lineshape)
    shapenumber = 4
    shapecomponent(4) = 6
    xcoordinate(4, 1) = 1
    ycoordinate(4, 1) = 0
    xcoordinate(4, 2) = 2
    ycoordinate(4, 2) = 0
    xcoordinate(4, 3) = 3
    ycoordinate(4, 3) = 0
    xcoordinate(4, 4) = 0
    ycoordinate(4, 4) = 1
    xcoordinate(4, 5) = 1
    ycoordinate(4, 5) = 1
    xcoordinate(4, 6) = 2
    ycoordinate(4, 6) = 1
End Sub
```

Adding New Preset 5

This preset is called a Glider. It is self-perpetuating and will progress across the screen in a south-east direction until it reaches the screen boundary. As shown in screenshot below the glider has four different states.



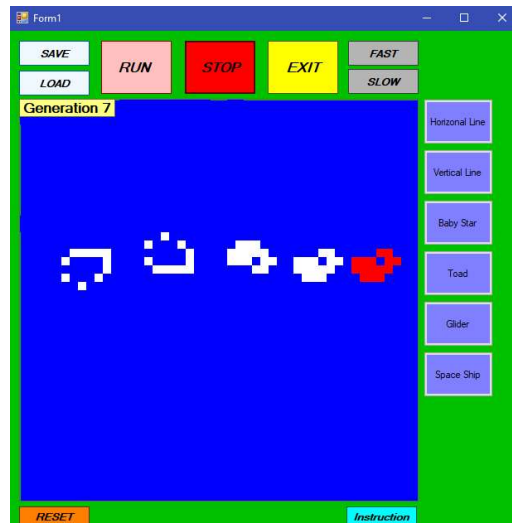
```
Private Sub cmdGlider_Click(sender As Object, e As EventArgs) Handles
cmdGlider.Click
```

```
    lineshape = Not (lineshape)
    shapenumber = 5
    shapecomponent(5) = 5
    xcoordinate(5, 1) = 0
    ycoordinate(5, 1) = 2
    xcoordinate(5, 2) = 1
    ycoordinate(5, 2) = 0
    xcoordinate(5, 3) = 1
    ycoordinate(5, 3) = 2
    xcoordinate(5, 4) = 2
    ycoordinate(5, 4) = 1
    xcoordinate(5, 5) = 2
    ycoordinate(5, 5) = 2
```

```
End Sub
```

Adding New Preset 6

This preset is called a medium sized spaceship. (There are both small and large spaceships as well). The medium sized space ship moves from right to left and has four states (shown in the screenshot below). It contains 15 pixels.



```

Private Sub cmdmws_Click(sender As Object, e As EventArgs) Handles cmdmws.Click
    lineshape = Not (lineshape)
    shapenumber = 6
    shapecomponent(6) = 15
    xcoordinate(6, 1) = 0
    ycoordinate(6, 1) = 1
    xcoordinate(6, 2) = 0
    ycoordinate(6, 2) = 2
    xcoordinate(6, 3) = 1
    ycoordinate(6, 3) = 1
    xcoordinate(6, 4) = 1
    ycoordinate(6, 4) = 2
    xcoordinate(6, 5) = 1
    ycoordinate(6, 5) = 3
    xcoordinate(6, 6) = 2
    ycoordinate(6, 6) = 1
    xcoordinate(6, 7) = 2
    ycoordinate(6, 7) = 2
    xcoordinate(6, 8) = 2
    ycoordinate(6, 8) = 3
    xcoordinate(6, 9) = 3
    ycoordinate(6, 9) = 0
    xcoordinate(6, 10) = 3
    ycoordinate(6, 10) = 2
    xcoordinate(6, 11) = 3
    ycoordinate(6, 11) = 3
    xcoordinate(6, 12) = 4
    ycoordinate(6, 12) = 0
    xcoordinate(6, 13) = 4
    ycoordinate(6, 13) = 1
    xcoordinate(6, 14) = 4
    ycoordinate(6, 14) = 2
    xcoordinate(6, 15) = 5
    ycoordinate(6, 15) = 1
End Sub

```

Object Orientated Code

The next stage of my development was to change my code to add object orientation to it. I decided to do this for several reasons. We had recently covered this topic at college and I was keen to see if I would be able to utilise this technique. Secondly, I wanted to demonstrate that I am able to program procedurally and object orientedly. Using OOP would make my code more readable and easier to maintain, and also it meant that I would be able to reduce my arrays from 2D to 1D and change 1D arrays to non dimensioned properties.

I began by defining a class:

```
Public Class ptive
    Public pname As String
    Public pshapenumber As Integer
    Public pshapecomponent As Integer
    Public pxcoordinate(20) As Integer
    Public pycoordinate(20) As Integer

End Class
```

I amended my routines to the updated variables "pshapenumber", "pshapecomponent", "pxcoordinate", "pycoordinate". For Example here is the routine that handles the mouse move event, but now contains the updated variables.

```
Private Sub pctCopy_MouseMove(sender As Object, e As MouseEventArgs)
    Dim a, b As Integer
    If lineshape = False Then
        Exit Sub
    End If
    ' outputpic = pctCopy.Image
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    For cntr = 1 To prim.pshapecomponent
        updatedisplay((olda + prim.pxcoordinate(cntr)), (oldb +
prim.pycoordinate(cntr)))
    Next
    olda = a
    oldb = b
    For cntr = 1 To prim.pshapecomponent
        mousedisplayred((olda + prim.pxcoordinate(cntr)), (oldb +
prim.pycoordinate(cntr)))
    Next
    ' pctCopy.Image = outputpic
End Sub
```

Next I amended the code associated with choosing a small primitive. For example here is the code that initiates the vertical line.

```

Private Sub cmdvline_Click(sender As Object, e As EventArgs) Handles
cmdvline.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 2
    prim.pshapecomponent = 3
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 0
    prim.pycoordinate(3) = 2
End Sub

```

Note that the arrays used to be two dimensional (pxcoordinate & pycoordinate) but are now one dimensional, and the one dimensional array e.g. pshape component is now a simple variable.

Before Object Orientation:

```

Private Sub cmdvline_Click(sender As Object, e As EventArgs) Handles
cmdvline.Click
    lineshape = Not (lineshape)
    shapenumber = 2
    shapecomponent(2) = 3
    xcoordinate(2, 1) = 0
    ycoordinate(2, 1) = 0
    xcoordinate(2, 2) = 0
    ycoordinate(2, 2) = 1
    xcoordinate(2, 3) = 0
    ycoordinate(2, 3) = 2
End Sub

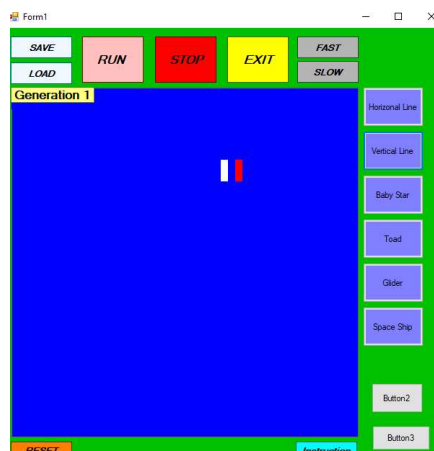
```

After Object Orientation:

```

Private Sub cmdvline_Click(sender As Object, e As EventArgs) Handles
cmdvline.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 2
    prim.pshapecomponent = 3
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 0
    prim.pycoordinate(3) = 2
End Sub

```



I then **tested** the updated code and it worked. (The two new buttons at the bottom of the screen were used to test and verify the new class).

```

Private Sub Button2_Click_1(sender As Object, e As EventArgs) Handles
Button2.Click
    ' test of oo
    *****
    *****

    prim.pprim.pshapenumber = 1
    prim.pprim.pshapecomponent = 4
    prim.pprim.pxcoordinate(1) = 0
    prim.pprim.pycoordinate(1) = 0
    prim.pprim.pxcoordinate(2) = 1
    prim.pprim.pycoordinate(2) = 0
    prim.pprim.pxcoordinate(3) = 2
    prim.pprim.pycoordinate(3) = 0
    prim.pprim.pxcoordinate(4) = 3
    prim.pprim.pycoordinate(4) = 0

    MsgBox(prim.pprim.pshapenumber)

End Sub

Private Sub Button3_Click_1(sender As Object, e As EventArgs) Handles
Button3.Click
    MsgBox(prim.pprim.pshapenumber)
End Sub

```

Once I had verified the method I amended the rest of my buttons as shown below.

```

Private Sub cmdvline_Click(sender As Object, e As EventArgs) Handles cmdvline.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 2
    prim.pshapecomponent = 3
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 0
    prim.pycoordinate(3) = 2
End Sub

Private Sub cmdbs_Click(sender As Object, e As EventArgs) Handles cmdbs.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 3
    prim.pshapecomponent = 5
    prim.pxcoordinate(1) = 1
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 1
    prim.pxcoordinate(4) = 2
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 2
End Sub

```



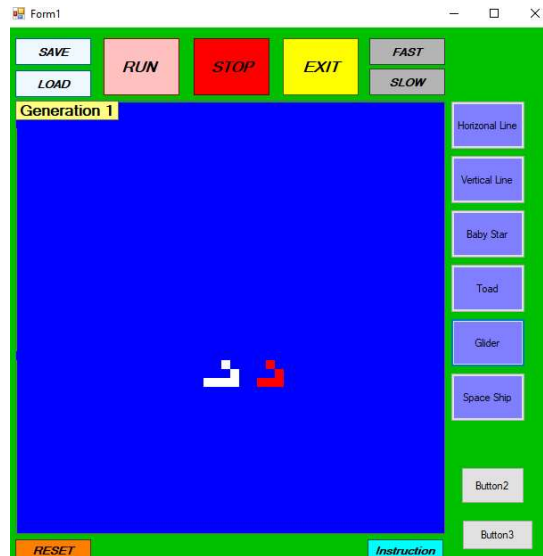
```
Private Sub cmdtoad_Click(sender As Object, e As EventArgs) Handles
cmdtoad.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 4
    prim.pshapecomponent = 6
    prim.pxcoordinate(1) = 1
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 2
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 3
    prim.pycoordinate(3) = 0
    prim.pxcoordinate(4) = 0
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 1
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
End Sub

Private Sub cmdGlider_Click(sender As Object, e As EventArgs) Handles
cmdGlider.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 5
    prim.pshapecomponent = 5
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 2
    prim.pxcoordinate(2) = 1
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 2
    prim.pxcoordinate(4) = 2
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 2
    prim.pycoordinate(5) = 2
End Sub

Private Sub cmdmws_Click(sender As Object, e As EventArgs) Handles cmdmws.Click
    lineshape = Not (lineshape)
    prim.pshapenumber = 6
    prim.pshapecomponent = 15
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 1
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 2
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 1
    prim.pxcoordinate(4) = 1
    prim.pycoordinate(4) = 2
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 3
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
    prim.pxcoordinate(7) = 2
    prim.pycoordinate(7) = 2
    prim.pxcoordinate(8) = 2
    prim.pycoordinate(8) = 3
    prim.pxcoordinate(9) = 3
    prim.pycoordinate(9) = 0
End Sub
```

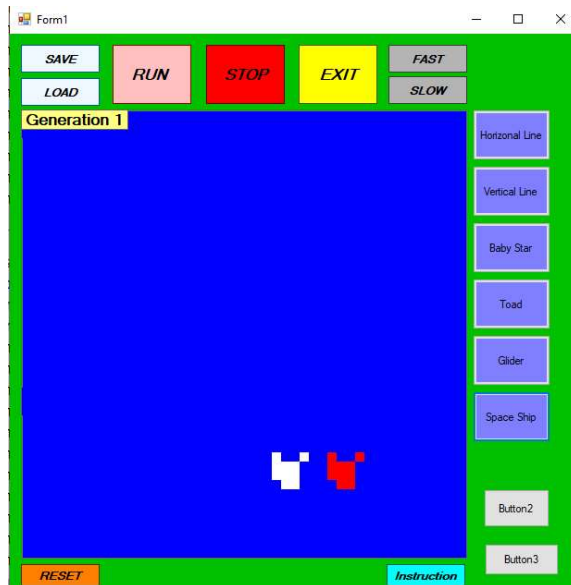
As I was amending my code I tested each procedure for functionality, here are screenshots two of the selected test:

Number 1 testing the gliders



As can be seen from this test both my mouse down and mouse move procedures are functioning correctly.

Number 2 testing the Spaceships

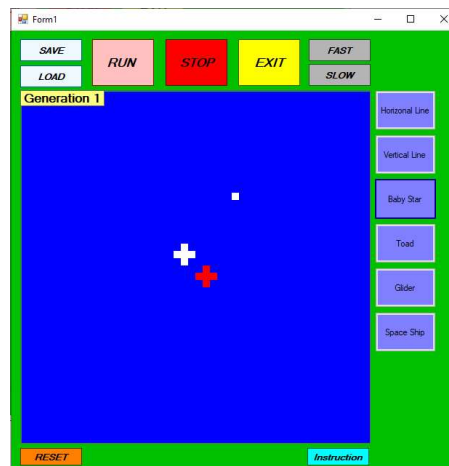


End User Testing

At this point I decided to show my nearly finished prototype to my end user (Mr Watson) for feedback. He was very pleased with the project so far but he had one thing that he asked me to improve. The way the project works at the moment is the user can select a primitive shape from one of the buttons on the right to add to the environment. The problem that my end user found with this was that in order to deselect the primitive shape the button itself had to be clicked a second time. Whilst he felt that this wasn't much of an issue for himself, he did feel that the students who were new to the software might initially find difficulty with this input method.

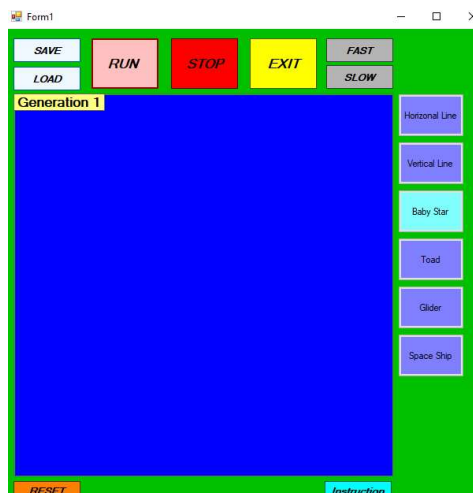
On reflection I agreed that with my end user that the input method was not fully intuitive so I embarked on further development in order to remedy the situation.

First thing I tried was to give a visual indication of which button had been selected in some way. I initially tried to change the appearance through the defaults such as sunken.



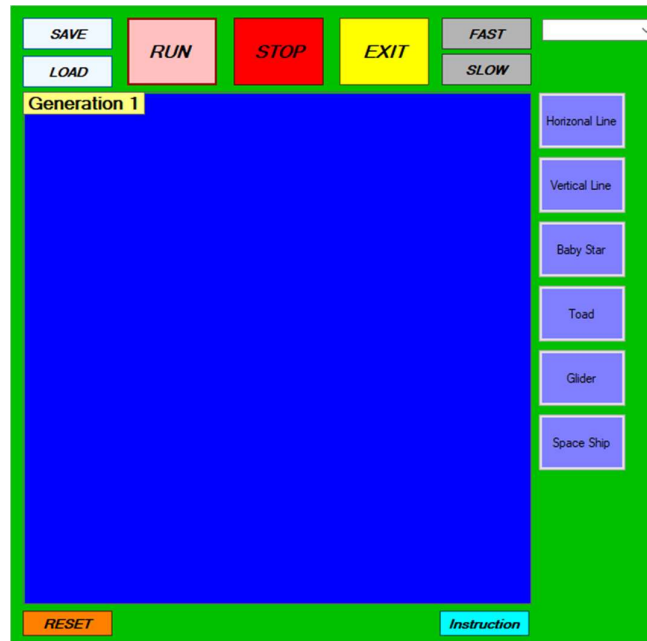
In this example "Baby Star" changes appearance once it has been clicked and then changes back once it is deselected. I thought this was quite a good solution but I showed my project to someone who had not seen it before and whilst they eventually understood the highlighting system, this understanding was not immediate. I felt that there must be a better solution.

Next I tried changing the whole colour and texture of the button.



Whilst this was a slight improvement there was still the issue that a user had to select and then deselect a button. Whilst I felt that I could include instruction on this I felt that actually the input method itself was at fault since it was only partially intuitive to a user. I therefore decided to try a different control.

After some research, I settled on a combo box. Here is my prototype:



```

Private Sub cmbChoice_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbChoice.SelectedIndexChanged
    Dim n As Integer
    n = cmbChoice.SelectedIndex
    MsgBox(n)
    If n = 0 Then
        lineshape = False
    End If
    If n = 1 Then
        lineshape = True
        prim.pshapenumber = 1
        prim.pshapecomponent = 4
        prim.pxcoordinate(1) = 0
        prim.pycoordinate(1) = 0
        prim.pxcoordinate(2) = 1
        prim.pycoordinate(2) = 0
        prim.pxcoordinate(3) = 2
        prim.pycoordinate(3) = 0
        prim.pxcoordinate(4) = 3
        prim.pycoordinate(4) = 0
    End If
End Sub

```

This seemed to work a lot better than my previous method as there was no ambiguity over whether a button has been "Selected" or not. I then showed my end user the technique and he was much happier with it and so I changed my solution to use a combo box for preset selection.

```
Private Sub cmbChoice_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cmbChoice.SelectedIndexChanged
    Dim n As Integer
    n = cmbChoice.SelectedIndex
    If n = 0 Then
        lineshape = False
    End If
    If n = 1 Then
        lineshape = True
        prim.pshapenumber = 1
        prim.pshapecomponent = 4
        prim.pxcoordinate(1) = 0
        prim.pycoordinate(1) = 0
        prim.pxcoordinate(2) = 1
        prim.pycoordinate(2) = 0
        prim.pxcoordinate(3) = 2
        prim.pycoordinate(3) = 0
        prim.pxcoordinate(4) = 3
        prim.pycoordinate(4) = 0
    End If

    If n = 2 Then
        lineshape = True
        prim.pshapenumber = 2
        prim.pshapecomponent = 3
        prim.pxcoordinate(1) = 0
        prim.pycoordinate(1) = 0
        prim.pxcoordinate(2) = 0
        prim.pycoordinate(2) = 1
        prim.pxcoordinate(3) = 0
        prim.pycoordinate(3) = 2
    End If

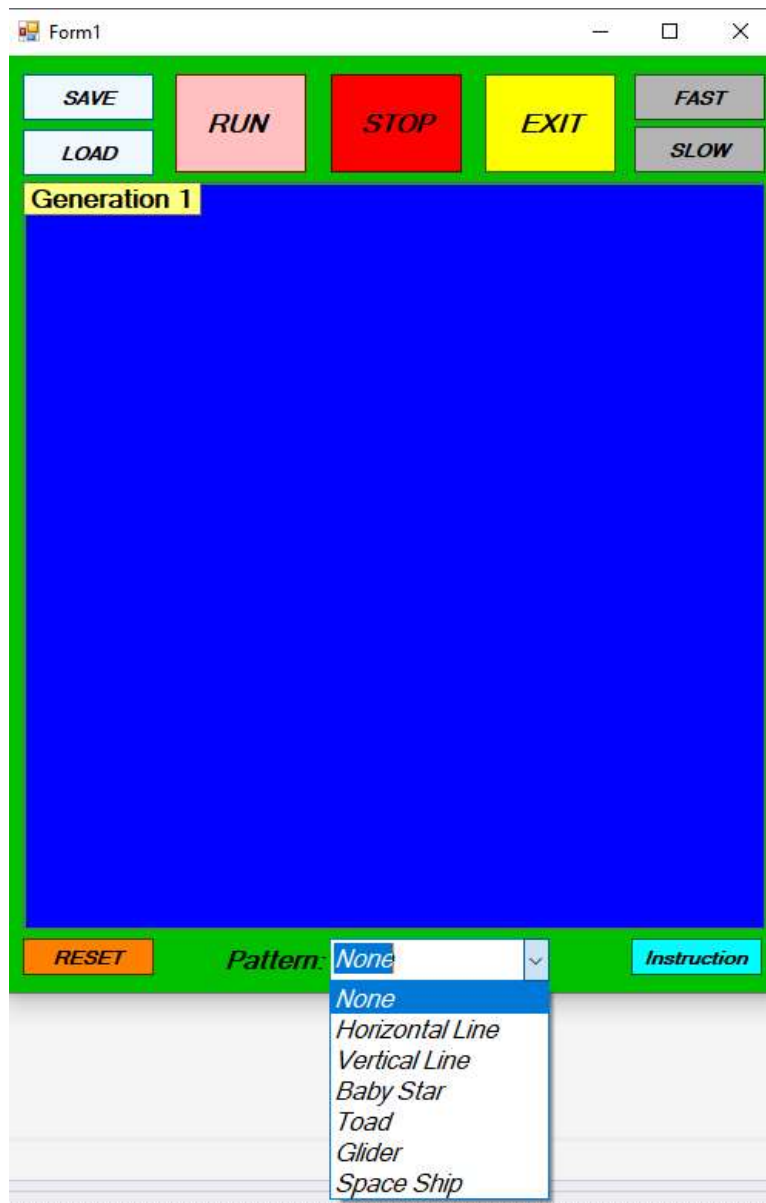
    If n = 3 Then
        lineshape = True
        prim.pshapenumber = 3
        prim.pshapecomponent = 5
        prim.pxcoordinate(1) = 1
        prim.pycoordinate(1) = 0
        prim.pxcoordinate(2) = 0
        prim.pycoordinate(2) = 1
        prim.pxcoordinate(3) = 1
        prim.pycoordinate(3) = 1
        prim.pxcoordinate(4) = 2
        prim.pycoordinate(4) = 1
        prim.pxcoordinate(5) = 1
        prim.pycoordinate(5) = 2
    End If
End Sub
```

```
If n = 4 Then
    lineshape = True
    prim.pshapenumber = 4
    prim.pshapecomponent = 6
    prim.pxcoordinate(1) = 1
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 2
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 3
    prim.pycoordinate(3) = 0
    prim.pxcoordinate(4) = 0
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 1
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
End If

If n = 5 Then
    lineshape = True
    prim.pshapenumber = 5
    prim.pshapecomponent = 5
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 2
    prim.pxcoordinate(2) = 1
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 2
    prim.pxcoordinate(4) = 2
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 2
    prim.pycoordinate(5) = 2
End If

If n = 6 Then
    lineshape = True
    prim.pshapenumber = 6
    prim.pshapecomponent = 15
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 1
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 2
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 1
    prim.pxcoordinate(4) = 1
    prim.pycoordinate(4) = 2
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 3
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
    prim.pxcoordinate(7) = 2
    prim.pycoordinate(7) = 2
    prim.pxcoordinate(8) = 2
    prim.pycoordinate(8) = 3
    prim.pxcoordinate(9) = 3
    prim.pycoordinate(9) = 0
End If
End Sub
```

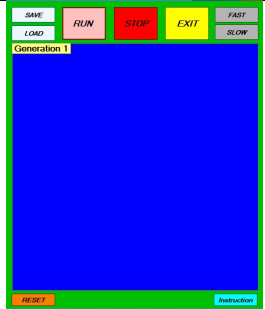
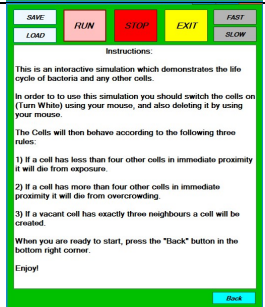

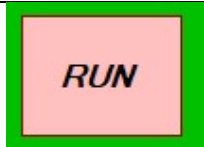

This the finished form:

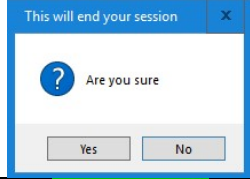

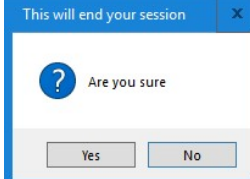
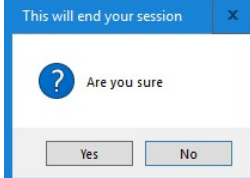







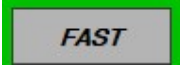
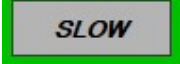
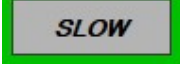
Testing

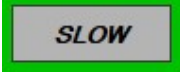
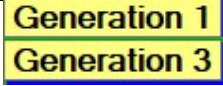
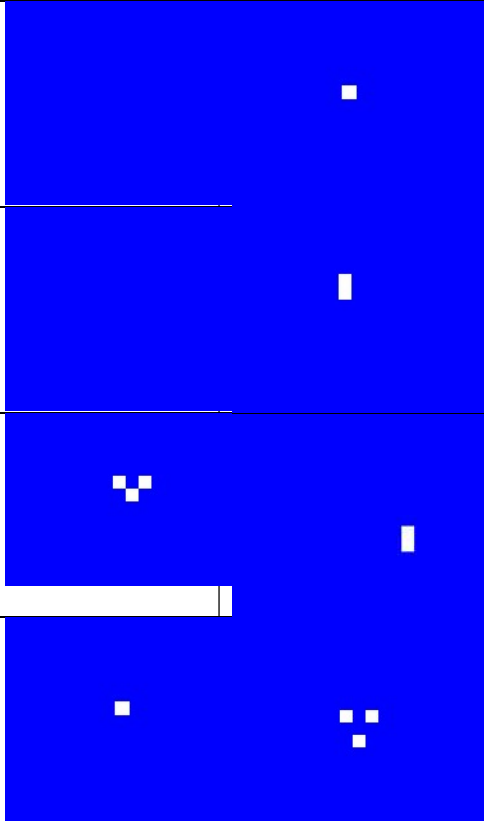




Here is my Testing Table.

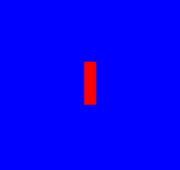
GUI Testing Table

Test No.	Test	Expected Results	Method / test data	Pass/Fail	Screenshot
1	Game initialisation	On Form Load the user is presented with a blank 50x50 grid with each element of the grid turned 'off'.	Run the program and observe the start up	Pass	
2	Make sure the instruction screen displays	The instructions are displayed on separate user interface	Click on instructions button view instructions	Pass	
3	Check Reset button is functional	User should be presented with a blank 50x50 grid.	The program will be run and stopped with a bit pattern in place before the reset button is pressed	Pass	
4	Check run button is functional	The program will begin calculating generations.	Program will be loaded and a bit pattern introduced to the user interface. The run button will then be clicked.	Pass	
5	Testing Stop button works	On clicking the stop button the generations should cease calculating.	I bit pattern will be introduced to the user interface before the run button is pressed. As the generation are being calculated the stop button will be clicked.	Fail but then moved the loop now Pass	

6	Check the Exit button message box	A message box should appear asking "are you sure you want to exit?"	The program is loaded and then the exit button is clicked.	Pass	
7	Check the Exit button works	A message box should appear asking "are you sure you want to exit?"	The program is loaded, but this time the program is run before being stopped and then the exit button is clicked.	Pass	
8	Testing the Yes/ No message box which follows exit click	On clicking the no option the message box should disappear and the computer should return to the program	The exit button is clicked followed by "no"	Pass	
9	Testing the Yes/ No message box which follows exit click	On clicking the yes option the code should stop running.	The exit button is clicked followed by "yes"	Pass	
10	Check Save button	On clicking save a copy of the current bit pattern should be saved to the user defined text file.	The program is loaded and a bit pattern created on the user interface. The save button is then pressed.	Pass	
11	Check Save button	On clicking save a copy of the current bit pattern should be saved to the user defined text file.	The program is loaded and a bit pattern created on the user interface. This time the program will calculate the generational iterations before being altered. The save button is then pressed.	Pass	
12	Check Load button	On clicking load a copy of a saved bit pattern should be displayed ready to be run.	A bit pattern will be introduced to the user interface and saved. The user interface will be reset and then the	Pass	

			saved bit pattern will be loaded. The loaded bit pattern should be the same as the saved bit pattern.		
13	Test Fast button	On clicking the fast button the time between generational iterations should decrease	The bit pattern will be introduced to the user interface and the run button pressed. The fast button will then be pressed which should speed up the program by half a second.	Pass	
14	Test fast button limit	On clicking the fast button multiple times the program should hit a speed limit of half a second per cycle.	Bit pattern will be introduced to the user interface and the run button selected. The fast button will then be clicked three times	Pass	
15	Test fast button limit	On clicking the fast button multiple times the program should hit a speed limit of half a second per cycle.	Bit pattern will be introduced to the user interface and the run button selected. The fast button will then be clicked four times	Pass	
16	Check slow button	On clicking the slow button the time between generational iterations should increase	The bit pattern will be introduced to the user interface and the run button pressed. The fast button will then be pressed which should slow down the program by half a second.	Pass	
17	Test slow button limit	On clicking the slow button multiple times the program should hit a speed limit of ten seconds per cycle.	Bit pattern will be introduced to the user interface and the run button selected. The slow button will then be clicked 16 times	Pass	

18	Test slow button limit	On clicking the slow button multiple times the program should hit a speed limit of 10 seconds per cycle.	Bit pattern will be introduced to the user interface and the run button selected. The slow button will then be clicked more than 16 times	Pass	
19	Test that the generation indicator is working	On running the program the number of generations should be displayed in generation text box	Bit pattern will be introduced to the user interface and the program started. As the generations increase so should the number.	Pass	
20	Test that if the cell has one neighbour it will disappear			Pass	
21	Test that if the cell has two neighbours is already off it will stay off			Pass	
22	Test that if the cell has two neighbours is already on it will stay on			Pass	
23	Test that if the cell has three neighbours it will grow			Pass	
24	Test that the pattern bar displays the shapes			On running the pattern bar the pre coded shaped are displayed	Click on the Pattern bar.

25	Test that the Horizontal line displays its ghost	the Horizontal line displays its ghost	Click on the horizontal bar the hover the mouse of the interface	Pass	
26	Test that the Vertical line displays its ghost	the Vertical line displays its ghost	Click on the vertical bar the hover the mouse of the interface	Pass	
27	Test that the Baby Star line displays its ghost	the Baby Star line displays its ghost	Click on the baby star bar the hover the mouse of the interface	Pass	
28	Test that the Toad line displays its ghost	the Toad line displays its ghost	Click on the Toad bar the hover the mouse of the interface	Pass	
29	Test that the Glider line displays its ghost	the Glider line displays its ghost	Click on the Glider bar the hover the mouse of the interface	Pass	
30	Test that the Space Ship line displays its ghost	the Space Ship line displays its ghost	Click on the Space Ship bar the hover the mouse of the interface	Pass	
31	Test that when clicked Horizontal is displayed	Horizontal cells is displayed	Click on the interface	Pass	
32	Test that when clicked Vertical is displayed	Vertical cells is displayed	Click on the interface	Pass	
33	Test that when clicked Baby Star is displayed	Baby Star cells is displayed	Click on the interface	Pass	
34	Test that when clicked Toad is displayed	Toad cells is displayed	Click on the interface	Pass	
35	Test that when clicked Glider is displayed	Glider cells is displayed	Click on the interface	Pass	
36	Test that when clicked Space Ship is displayed	Space Ship cells is displayed	Click on the interface	Pass	

Beta Testing

To make sure that my project is up to the requirements I have made, I showed my project to Mr Watson and a selected group of year 11s, I asked them the following questionnaire.

To test my project was:

		Not Very					Very
		1	2	3	4	5	
1	To what extent do you think this computer simulation looks a living thing						
2	To what extent do you think the colours help the user to understand the concept						
3	How easy did you find the interface to use?						
4	Has this helped you understand the concept of environmental factors in the growth of an organism						
5	Any Suggestions for improvement						
6	Any other comments?						

Here is what Mr Watson said about my project:

		Mr Watson
1	To what extent do you think this computer simulation looks a living thing	5
2	To what extent do you think the colours help the user to understand the concept	5
3	How easy did you find the interface to use?	5
4	Has this helped your students understand the concept of environmental factors in the growth of an organism	5
5	Any Suggestions for improvement	No its perfect just what I wanted
6	Any other comments?	Thank You Very Much for doing this

Here are the student responses:

		Respondent 1	Respondent 2	Respondent 3	Respondent 4	Respondent 5	Average
1	To what extent do you think this computer simulation looks a living thing	4	5	3	3	4	3.8
2	To what extent do you think the colours help the user to understand the concept	5	4	5	5	4	4.6
3	How easy did you find the interface to use?	5	4	3	4	4	4
4	Has this helped you understand the concept of environmental factors in the growth of an organism	3	4	3	3	3	3.2
5	Any Suggestions for improvement			Would be better in yellow			
6	Any other comments?	No, its really cool				I like it a lot	

Beta Testing Result

Overall I would judge this success criteria to be met. As you can see from the above responses Mr Watson assessed my project at 5 out of 5 for realism. A select group of 5 year 11 students who Mr Watson showed my project to give an average score of 3.8. Even though this last score is lower than I would have hoped it is still over half or indeed better than 60% so I am happy with this. If I were to develop the solution further it would be interesting to work with a group of students on improving this point, but as I wasn't allowed to talk to students directly I wasn't able to do that this time.

Evalutaion

In this section I will evalate my solution.

What went well:

Although I found this project challenging I am glad that I completed it as I have substantially improved my programing knowledge. One of the main things that I am pleased with is that I taught myself visual basic forms. This is significantly more difficult than the console application, as there are countless menus, sub menus, forms and properties to master. In fact, it probably took me as long to learn the language as it did to do the project. As you will see from earlier in my project I was able to program a simplified (non-animated) version of my project in console relatively quickly.

A further thing that I enjoyed was learning about the game of life. It is interesting to know that this idea was first conceived of by John Conway in 1970. I really do think that the patterns the simulation produces mimic basic life, and it is very exciting to create something that growths in unexpected ways.

End User Sign Off

My end user was very pleased with the project and accepted the final version. Here is an email from him.

Feedback/ Evidence



Mr. Watson

Mon 02/03/2020 11:20

To: Jonathan G Maud (180154);

Dear Jonathan Maud

I am very impressed with the project. It worked to the high standards needed and the students loved using it.

The students got that needed it always asked to use this project when going over the life cycle of the cell, and with the interface of the project being aesthetically pleasing as well as allowing it to be flexible and easy to use I can see why it is so popular.

The formatting is clear, colourful and simple, which made it easier for everyone in my class to use.

To be honest I was expecting the project to be glitchy was so captivated by how robust and stable the project operated when in use. It was easy to change the parameters to suit the students tastes and own level which made it even better to use.

Very well done, the project is a perfect addition to my teaching tools and will definitely be using it in the years to come.

Kind Regards

Mr Watson

Evaluation of Success Criteria

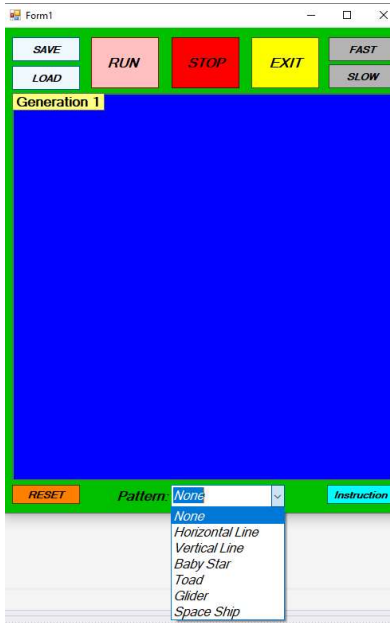
In this section I will evaluate my solution to see how closely it matches my success criteria. As you can see from the design section my success criteria are as follows:

- 1) *My project's user interface must be welcoming and easy to use in the classroom environment.*
- 2) *My Solution should mimic the life cycle of cells in an artificial habitat, using the rules from Conway's Game of Life.* I will measure this criteria using feedback from the year 11 students and Mr Watson:
- 3) *Create a fully functioning animation*
- 4) *The solution that I provide must be better at teaching the year 11 students the life cycle of Conway's cells as the other teaching methods* (see research), I will measure this based on feedback from the teacher Mr Watson and the year 11 students.
- 5) *My project must be able to store user created designs*
- 6) *My solution must be robust and should not crash when being used* as this would disrupt the lesson, resulting in wasted time and ruining the classroom atmosphere.
- 7) *My solution should be fully documented which will facilitate further development either myself or by a 3rd party should this be required in the future.*
- 8) *My solution should engage students*, I will judge these criteria based on the feedback from my end user (Mr Watson) and the Year 11 students.
- 9) *My solution must be easy to use.* I will judge this criterion based on feedback from my end user (Mr Watson) and the Year 11 students.
- 10) *My solution should take up a limited amount of space on a hard drive. I am aiming for a maximum footprint of 30mb.*
- 11) *My project must be rendered effectively using the school's workstations.* This will mean that all the students will be able to use the simulation without any performance issues.
- 12) *My solution should quick to load, I would estimate the longest time the end user (Mr Watson) and the students will have to wait would be maximum 10 seconds.*
- 13) *My Project should generate the cells by following the set rules*, regardless of the pattern or shape created by the user.
- 14) *Store the designs created by the user*, so that the student will be able to carry on where they left off last time they used the project.
- 15) *My solution must run quickly enough to create persistence of vision.*
- 16) *My solution must be capable of running on the minimum specification* computer that my end user (Mr Watson) might encounter.
- 17) *My solution must be capable of running on the school's network*, due to it having a numerous number of security protocols.

I will now evaluate each criterion in turn to see how successful my project has been.

- 1) *My project's user interface must be welcoming and to be used easily in the classroom environment.*

I am very pleased with my user interface:



This is actually the first time that I have created a graphical user interface with working buttons. My previous coding experience was one console app and the difference in end product is considerable. Whilst I couldn't imagine an end user who was low skilled using my solution that was console based, I feel that I could train anyone to use the interface of my finished product. Also I have designed the interface to be intuitive so that a person with a responsible degree of computer literacy would be able to use it with limited or no instruction. As you can see from the development section my end user had significant input into the design of the interface and I feel that this has really stretched my skill set and improved the final product a lot. Overall I would say that this success criteria are met.

- 2) *My Solution should mimic the life cycle of cells in an artificial habitat, using the rules from Conway's Game of Life.*

As seen in the Beta Testing section. I said that this success criteria would be assessed by my end-users. Once my project was complete I showed it to Mr Watson and a selected group of year 11s, I asked them the following questionnaire:

		Not				Very
		1	2	3	4	5
1	To what extent do you think this computer simulation looks a living thing					
2	To what extent do you think the colours help the user to understand the concept					
3	How easy did you find the interface to use?					
4	Has this helped you understand the concept of environmental factors in the growth of an organism					
5	Any Suggestions for improvement					
6	Any other comments?					

Here is what Mr Watson said about my project:

		Mr Watson
1	To what extent do you think this computer simulation looks a living thing	5
2	To what extent do you think the colours help the user to understand the concept	5
3	How easy did you find the interface to use?	5
4	Has this helped your students understand the concept of environmental factors in the growth of an organism	5
5	Any Suggestions for improvement	No its perfect just what I wanted
6	Any other comments?	Thank You Very Much for doing this

Here are the student responses:

		Respondent 1	Respondent 2	Respondent 3	Respondent 4	Respondent 5	Average
1	To what extent do you think this computer simulation looks a living thing	4	5	3	3	4	3.8
2	To what extent do you think the colours help the user to understand the concept	5	4	5	5	4	4.6
3	How easy did you find the interface to use?	5	4	3	4	4	4
4	Has this helped you understand the concept of environmental factors in the growth of an organism	3	4	3	3	3	3.2
5	Any Suggestions for improvement			Would be better in yellow			
6	Any other comments?	No, its really cool				I like it a lot	

Overall I would judge this success criteria to be met. As you can see from the above responses Mr Watson assessed my project at 5 out of 5 for realism. A select group of 5 year 11 students who Mr Watson showed my project to give an average score of 3.8. Even though this last score is lower than I would have hoped it is still over half or indeed better than 60% so I am happy with this. If I were to develop the solution further it would be interesting to work with a group of students on improving this point, but as I wasn't allowed to talk to students directly I wasn't able to do that this time.

3) *Create a fully functioning simulation*

As can be seen from my testing section and also my end user signoff the solution is fully functional so I would say that this criterion has been met.

- 4) *The solution that I provide must be better at teaching the year 11 students the life cycle of Conway's cells as the other teaching methods*

This point I am not so sure of and having completed the project I feel that the criteria was a little too ambitious. As you can see from the above questionnaire students rated my program as a 3.2 out of 5 for "helping you understand the concept of environmental factors in the growth of an organism?" Whilst pleased with this as a result, my solution does not include any of the theory that the students need to know about the topic so realistically can only form part of Mr Watson's strategy for teaching this topic. Overall I would say this success criteria were not met.

- 5) *My project must be able to store the design created by the student*

As can be seen from the development section my project has a function which will allow a user defined pattern to be recorded and reloaded, and so I will judge this criterion to have been well met.

- 6) *My solution must be robust and should not crash when being used*

Throughout my formal testing and also my informal use my solution never showed any signs of hanging or exiting in an unexpected manner. It has been used by several test users and none of those reported any stability issues either, therefore I would judge this criterion to be well met.

- 7) *My solution should be fully documented which will facilitate further development either myself or by a 3rd party should this be required in the future.*

Although I commented my code, I have not provided either a list of variables used or a list of functions so I would say that this criterion is only partially met.

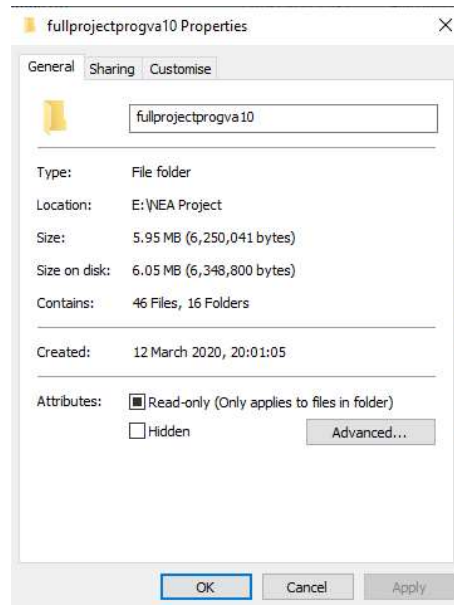
- 8) *My solution should engage students*

Anecdotally Mr Watson told me that the student who tested my project liked it. This can be evidence by looking at the scores for the questionnaire which were all better than 50% so I would say that this criterion has been met.

- 9) *My solution must be easy to use.*

All the students including Mr Watson all said that my project was easy to use and as a result the success criteria was met.

- 10) *My solution should take up a limited amount of space on a hard drive. I am aiming for a maximum footprint of 30mb.*



As may be seen from the above screenshot my finished solution had a footprint of 6.05MB so this criterion has been met.

- 11) *My solution should quick to load, I would estimate the longest time the end user (Mr Watson) and the students will have to wait would be maximum 10 seconds.*

At the moment my project takes over 40 seconds to load into visual studio however if I was to compile my code into binary it would take much less time as a result. This is one of the things that I would like to do if I were to continue with this project.

- 12) *My Project should generate the cells by following the set rules*

As can be seen from the testing section this criterion has been met.

- 13) *Store the designs created by the user*

As can be seen from the development section my project has a function which will allow a user defined pattern to be recorded and reloaded, and so I will judge this criterion to have been well met.

- 14) *My solution must run quickly enough to create persistence of vision.*

In the end my solution didn't meet this criterion. However, I believe this modification that I mention later in this section using 1s and 0s instead of Os and Xs would have a significant uplift in performance.

- 15) *My solution must be capable of running on the school's network*

This criteria was met.

What I would change if I was to do this again:

Future development

When I was part of the way through my project I realised that I could improve its efficiency by changing the underlying data structure. It was little too late in my schedule to change before the project deadline so I am mentioning here as a thought for future development. When I started the project I had to decide on a data structure to represent each pixel's property (on/off). I decided at the time to use an "x" for on and a "o" for off. This was a reasonably arbitrary decision at the time although I chose the data structure as it was easier for me to understand. I had recently been practicing with noughts and crosses programs prior to starting the game of life and I carried this data structure over. I had heard of Booleans, which would probably have been more efficient from a data usage point of view (one bit per pixel compared to one bite per pixel), but at the time Booleans were new to me and so I stuck with what I knew. If I were to do this again I would investigate the use of Boolean variables to store my data.

A further possibility for improving my code would be to use an integer to store the pixel information instead of "x" and "o". Although this would be less efficient from a memory usage point of view compared to a Boolean, I could make significant efficiency savings in my coding. The existing code which calculates each points number of neighbours looks like this:

```

For counterx = 1 To maxx
  For country = 1 To maxy
    If pond((counterx - 1), (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx - 1), country) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx - 1), (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond(counterx, (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond(counterx, (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), (country - 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), country) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
    If pond((counterx + 1), (country + 1)) = "x" Then
      neighbours(counterx, country) = neighbours(counterx, country) + 1
    End If
  Next
Next

```

If I had used a one and a zero to represent my data I could have replaced this with a simple addition calculation, i.e.

Neighbours (counter x, counter y) = pond (counter x - 1, counter y - 1) + pond (counter x - 1, counter y) + pond (counter x - 1, counter y + 1) + pond (counter x, counter y - 1) + pond (counter x, counter y) + pond (counter x, counter y + 1) + pond (counter x + 1, counter y - 1) + pond (counter x + 1, counter y) + pond (counter x + 1, counter y + 1)

It most likely that this would have executed much quicker than the way that I coded it in my solution (8 if statements). As this calculation needs to be performed 2, 500 times per generation I believe that this improvement would significantly improve the execution time of my code. If I were to do this project again, or update or maintain this project, I would look into the possibility of changing the data structure.

Conclusion

Notwithstanding the developments needed to be added to my project, my project still meets the end user's/ Mr Watson's criteria. I have learnt a lot about coding, cells, Conway's theory and learning how to use forms, the transition from console to forms was a challenge, however after not to long I adapted and was able to produce a professional looking project that fitted the criteria needed. After going through my feedback I can conclude that my project has been a success and I am very happy with how the project turned out and I truly believe that my project is a unique learning platform for students studying the life cycle of a cell according to Conway's theory for Game of Life.

Appendix: Full Code Listing

```
Imports System.IO
Public Class Form1
    'Initialise Variables
    Dim outputpic As New Bitmap(520, 520,
System.Drawing.Imaging.PixelFormat.Format32bppArgb)
    Dim pond(51, 51) As String
    Dim maxx As Integer = 50
    Dim maxy As Integer = 50
    Dim outputline As String
    Dim neighbours(maxx, maxy) As Integer
    Dim nextgen(maxx, maxy) As String
    Dim generation As Integer = 1
    Dim blocksize As Integer = 9
    Dim olda, oldb As Integer
    Dim lineshape As Boolean = False
    Dim oldcolour(20) As String
    Dim shapecomponent(20) As Integer
    Dim xcoordinate(20, 20) As Integer
    Dim ycoordinate(20, 20) As Integer
    Dim shapenumber As Integer
    Dim prim As New ptive
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        'Set the Clock Ticking
        Tmrtimer.Enabled = True
    End Sub
    Private Sub pctDisplay_MouseDown(sender As Object, e As MouseEventArgs) Handles
pctDisplay.MouseDown
        'This handles mouse click
        Dim a As Integer
        Dim b As Integer
        Dim blocksize As Integer = 9
        Dim counterx, countery As Integer
        a = (e.X.ToString)
        b = (e.Y.ToString)
        a = Int(a / 10)
        b = Int(b / 10)
        'has a primitive shape been selected
        If lineshape = True Then
            For cntr = 1 To prim.pshapecomponent
                pond((a + prim.pxcoordinate(cntr)), (b + prim.pycoordinate(cntr))) = "x"
            Next
            For cntr = 1 To prim.pshapecomponent
                mousedisplaywhite((olda + prim.pxcoordinate(cntr)), (oldb +
prim.pycoordinate(cntr)))
            Next
            'if not primitive selected
        Else
            If pond(a, b) = "x" Then
                pond(a, b) = "O"
            Else
                pond(a, b) = "x"
            End If
            For x = (a * 10) To (a * 10 + 9)
                For y = (b * 10) To (b * 10 + 9)
                    If pond(a, b) = "x" Then
                        outputpic.SetPixel(x, y, Color.White)
                    Else
                        outputpic.SetPixel(x, y, Color.Blue)
                    End If
                Next
            Next
            End If
            pctDisplay.Image = outputpic
        End Sub
End Sub
```

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'setting up the form and array
    For counterx = 1 To maxx
        For country = 1 To maxy
            pond(counterx, country) = "0"
        Next
    Next
    For k = 1 To 500
        For l = 1 To 500
            outputpic.SetPixel(k, l, Color.Blue)
        Next
    Next
    pctDisplay.Image = outputpic
    For x = 1 To maxx
        For y = 1 To maxy
            neighbours(x, y) = 0
        Next
    Next
    Tmrtimer.Enabled = False
End Sub
Private Sub cmdInstructions_Click(sender As Object, e As EventArgs) Handles
cmdInstructions.Click
    'Click to show insructions
    Form2.Show()
    Me.Hide()
End Sub
Private Sub Tmrtimer_Tick(sender As Object, e As EventArgs) Handles Tmrtimer.Tick
    'Every tick of the clock update generation
    Dim clr As New Color
    'count neighbours
    For counterx = 1 To maxx
        For country = 1 To maxy
            If pond((counterx - 1), (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx - 1), country) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx - 1), (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond(counterx, (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond(counterx, (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), (country - 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), country) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
            If pond((counterx + 1), (country + 1)) = "x" Then
                neighbours(counterx, country) = neighbours(counterx, country) + 1
            End If
        Next
    Next
    lblGeneration.Text = ("Generation " & generation)

```



```

'calculate growth or decline of each cell
For counterx = 1 To maxx
  For country = 1 To maxy
    'decline
    If neighbours(counterx, country) < 2 Or neighbours(counterx, country) > 3 Then
      nextgen(counterx, country) = "0"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.Blue)
        Next
      Next
    End If
    'growth
    If neighbours(counterx, country) = 3 Then
      nextgen(counterx, country) = "x"
      For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
        For subcountry = (10 * country) To (10 * country + blocksize)
          outputpic.SetPixel(subcounterx, subcountry, Color.White)
        Next
      Next
    End If
    'static
    If neighbours(counterx, country) = 2 Then
      If pond(counterx, country) = "x" Then
        nextgen(counterx, country) = "x"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country + blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.White)
          Next
        Next
      Else
        nextgen(counterx, country) = "0"
        For subcounterx = (10 * counterx) To (10 * counterx + blocksize)
          For subcountry = (10 * country) To (10 * country + blocksize)
            outputpic.SetPixel(subcounterx, subcountry, Color.Blue)
          Next
        Next
      End If
    End If
  Next
Next
'reset neighbours array
For counterx = 1 To maxx
  For country = 1 To maxy
    pond(counterx, country) = nextgen(counterx, country)
    neighbours(counterx, country) = 0
  Next
Next
'update display
generation = generation + 1
pctDisplay.Image = outputpic
Me.Refresh()
End Sub
Private Sub cmdEXIT_Click(sender As Object, e As EventArgs) Handles cmdEXIT.Click
  'exit software
  Dim answer As Integer
  answer = MsgBox("Are you sure", vbQuestion + vbYesNo + vbDefaultButton2, "This will end
your session")
  If answer = vbYes Then
    Me.Close()
  End If
End Sub
Private Sub cmdStop_Click(sender As Object, e As EventArgs) Handles cmdStop.Click
  'halt generations
  Tmrtimer.Enabled = False
End Sub
Private Sub cmdspeedup_Click(sender As Object, e As EventArgs) Handles cmdspeedup.Click

```

```

Private Sub cmdspeedup_Click(sender As Object, e As EventArgs) Handles cmdspeedup.Click
    'speed up simulation
    Dim timeinterval As Integer
    timeinterval = Tmrtimer.Interval
    timeinterval = timeinterval - 500
    If timeinterval < 500 Then
        timeinterval = 500
    End If
    Tmrtimer.Interval = timeinterval
End Sub
Private Sub cmdSLOWDOWN_Click(sender As Object, e As EventArgs) Handles
cmdSLOWDOWN.Click
    'slow down simulation
    Dim timeinterval As Integer
    timeinterval = Tmrtimer.Interval
    timeinterval = timeinterval + 500
    If timeinterval > 10000 Then
        timeinterval = 10000
    End If
    Tmrtimer.Interval = timeinterval
End Sub
Private Sub cmdsave_Click(sender As Object, e As EventArgs) Handles cmdsave.Click
    'save current pattern
    Dim fname As String
    fname = InputBox("What Would You Like to Call it?")
    fname = fname & ".txt"
    Dim filewrite As New StreamWriter(fname)
    For xcount = 0 To 50
        For ycount = 0 To 50
            filewrite.WriteLine(pond(xcount, ycount))
        Next
    Next
    filewrite.Close()
End Sub
Private Sub cmdLOAD_Click(sender As Object, e As EventArgs) Handles cmdLOAD.Click
    'load a preprepared pattern from file
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, countery As Integer
    Dim fname As String
    fname = InputBox("Which File Would You Like To Load?")
    fname = fname & ".txt"
    Dim fileread As StreamReader = New StreamReader(fname)
    For xcount = 0 To 50
        For ycount = 0 To 50
            pond(xcount, ycount) = fileread.ReadLine
        Next
    Next
    fileread.Close()
    Dim outputline As String
    For xcount = 0 To 50
        For ycount = 0 To 50
            outputline = outputline & pond(xcount, ycount)
        Next
    Next
    For a = 0 To 50
        For b = 0 To 50
            For x = (a * 10) To (a * 10 + 9)
                For y = (b * 10) To (b * 10 + 9)
                    If pond(a, b) = "x" Then
                        outputpic.SetPixel(x, y, Color.White)
                    Else
                        outputpic.SetPixel(x, y, Color.Blue)
                    End If
                Next
            Next
        Next
    Next
Next

```

```

pctDisplay.Image = outputpic
End Sub
Private Sub reset_Click(sender As Object, e As EventArgs) Handles reset.Click
    'reset display
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, countery As Integer
    For a = 0 To 50
        For b = 0 To 50
            For x = (a * 10) To (a * 10 + 9)
                For y = (b * 10) To (b * 10 + 9)
                    pond(a, b) = "o"
                    outputpic.SetPixel(x, y, Color.Blue)
                Next
            Next
        Next
    Next
    pctDisplay.Image = outputpic
    generation = 1
    lblGeneration.Text = "Generation " & generation
End Sub
Private Sub pctDisplay_MouseMove(sender As Object, e As MouseEventArgs) Handles
pctDisplay.MouseMove
    'update cursor trail
    Dim a, b As Integer
    'no primitive selected
    If lineshape = False Then
        pctDisplay.Image = outputpic
        Exit Sub
    End If
    'if primitive selected
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    For cntr = 1 To prim.pshapecomponent
        updatedisplay((olda + prim.pxcoordinate(cntr)), (oldb +
prim.pycoordinate(cntr)))
    Next
    olda = a
    oldb = b
    For cntr = 1 To prim.pshapecomponent
        mousedisplayred((olda + prim.pxcoordinate(cntr)), (oldb +
prim.pycoordinate(cntr)))
    Next
    pctDisplay.Image = outputpic
    pctDisplay.Image = outputpic
End Sub
Public Sub mousedisplayred(dispa As Integer, dispb As Integer)
    'plot red mouse trail
    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            outputpic.SetPixel(x, y, Color.Red)
        Next
    Next
End Sub
Public Sub mousedisplayblue(dispa As Integer, dispb As Integer)
    'reset mouse trail
    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            outputpic.SetPixel(x, y, Color.Blue)
        Next
    Next
End Sub

```

```

Private Sub pctDisplay_MouseLeave(sender As Object, e As EventArgs) Handles
pctDisplay.MouseLeave
    'when mouse leaves display reset trail
    mousedisplayblue(oldda, oldb)
    mousedisplayblue((oldda + 1), oldb)
    mousedisplayblue((oldda + 2), oldb)
    pctDisplay.Image = outputpic
End Sub
Private Sub cmdShapeLine_Click(sender As Object, e As EventArgs)
    'primitive shape horizontal line
    Dim a, b As Integer
    lineshape = Not (lineshape)
    shapenumber = 1
    shapecomponent(1) = 3
    xcoordinate(1, 1) = 0
    ycoordinate(1, 1) = 0
    xcoordinate(1, 2) = 1
    ycoordinate(1, 2) = 0
    xcoordinate(1, 3) = 2
    ycoordinate(1, 3) = 0
End Sub
Private Sub Button3_Click(sender As Object, e As EventArgs)
    'button click update
    For counterx = 1 To maxx
        For countery = 1 To maxy
            pond(counterx, countery) = "0"
        Next
    Next
    For k = 1 To 500
        For l = 1 To 500
            outputpic.SetPixel(k, l, Color.Blue)
        Next
    Next
    For x = 1 To maxx
        For y = 1 To maxy
            neighbours(x, y) = 0
        Next
    Next
End Sub
Private Sub pctCopy_MouseDown(sender As Object, e As MouseEventArgs)
    'copy image on mouse down
    Dim a As Integer
    Dim b As Integer
    Dim blocksize As Integer = 9
    Dim counterx, countery As Integer
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    If lineshape = True Then
        For cntr = 1 To shapecomponent(shapenumber)
            pond((a + xcoordinate(shapenumber, cntr)), (b +
            ycoordinate(shapenumber, cntr))) = "x"
        Next
    End If
End Sub

```

```

Private Sub pctCopy_MouseMove(sender As Object, e As MouseEventArgs)
    'copy image when mouse move
    Dim a, b As Integer
    If lineshape = False Then
        Exit Sub
    End If
    a = (e.X.ToString)
    b = (e.Y.ToString)
    a = Int(a / 10)
    b = Int(b / 10)
    For cntr = 1 To shapecomponent(shapenumber)
        updatedisplay((olda + xcoordinate(shapenumber, cntr)), (oldb +
ycoordinate(shapenumber, cntr)))
    Next
    olda = a
    oldb = b
    For cntr = 1 To shapecomponent(shapenumber)
        mousedisplayred((olda + xcoordinate(shapenumber, cntr)), (oldb +
ycoordinate(shapenumber, cntr)))
    Next
End Sub
Public Sub mousedisplaywhite(dispa As Integer, dispb As Integer)
    'add new pixel to display
    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            outputpic.SetPixel(x, y, Color.White)
        Next
    Next
End Sub
Private Sub pctCopy_MouseLeave(sender As Object, e As EventArgs)
    'copy image when mouse leave
    mousedisplayblue(olda, oldb)
End Sub
Private Sub Button5_Click(sender As Object, e As EventArgs)
    'primitive shape horizontal line
    lineshape = Not (lineshape)
    shapenumber = 1
    shapecomponent(1) = 4
    xcoordinate(1, 1) = 0
    ycoordinate(1, 1) = 0
    xcoordinate(1, 2) = 1
    ycoordinate(1, 2) = 0
    xcoordinate(1, 3) = 2
    ycoordinate(1, 3) = 0
    xcoordinate(1, 4) = 3
    ycoordinate(1, 4) = 0
End Sub
Private Sub cmdvline_Click(sender As Object, e As EventArgs)
    'primitive shape vertical line
    lineshape = Not (lineshape)
    shapenumber = 2
    shapecomponent(2) = 3
    xcoordinate(2, 1) = 0
    ycoordinate(2, 1) = 0
    xcoordinate(2, 2) = 0
    ycoordinate(2, 2) = 1
    xcoordinate(2, 3) = 0
    ycoordinate(2, 3) = 2
End Sub

```

```
Private Sub cmdbs_Click(sender As Object, e As EventArgs)
    'primitive shape Baby Star
    lineshape = Not (lineshape)
    shapenumber = 3
    shapecomponent(3) = 5
    xcoordinate(3, 1) = 1
    ycoordinate(3, 1) = 0
    xcoordinate(3, 2) = 0
    ycoordinate(3, 2) = 1
    xcoordinate(3, 3) = 1
    ycoordinate(3, 3) = 1
    xcoordinate(3, 4) = 2
    ycoordinate(3, 4) = 1
    xcoordinate(3, 5) = 1
    ycoordinate(3, 5) = 2
End Sub
Private Sub cmdtoad_Click(sender As Object, e As EventArgs)
    lineshape = Not (lineshape)
    'primitive shape Toad
    shapenumber = 4
    shapecomponent(4) = 6
    xcoordinate(4, 1) = 1
    ycoordinate(4, 1) = 0
    xcoordinate(4, 2) = 2
    ycoordinate(4, 2) = 0
    xcoordinate(4, 3) = 3
    ycoordinate(4, 3) = 0
    xcoordinate(4, 4) = 0
    ycoordinate(4, 4) = 1
    xcoordinate(4, 5) = 1
    ycoordinate(4, 5) = 1
    xcoordinate(4, 6) = 2
    ycoordinate(4, 6) = 1
End Sub
Private Sub cmdglider_Click(sender As Object, e As EventArgs)
    lineshape = Not (lineshape)
    'primitive shape Glider
    shapenumber = 5
    shapecomponent(5) = 5
    xcoordinate(5, 1) = 0
    ycoordinate(5, 1) = 2
    xcoordinate(5, 2) = 1
    ycoordinate(5, 2) = 0
    xcoordinate(5, 3) = 1
    ycoordinate(5, 3) = 2
    xcoordinate(5, 4) = 2
    ycoordinate(5, 4) = 1
    xcoordinate(5, 5) = 2
    ycoordinate(5, 5) = 2
End Sub
Private Sub cmdmws_Click(sender As Object, e As EventArgs)
    lineshape = Not (lineshape)
    'primitive shape Medium Space ship
    shapenumber = 6
    shapecomponent(6) = 15
    xcoordinate(6, 1) = 0
    ycoordinate(6, 1) = 1
    xcoordinate(6, 2) = 0
    ycoordinate(6, 2) = 2
    xcoordinate(6, 3) = 1
    ycoordinate(6, 3) = 1
    xcoordinate(6, 4) = 1
    ycoordinate(6, 4) = 2
    xcoordinate(6, 5) = 1
    ycoordinate(6, 5) = 3
    xcoordinate(6, 6) = 2
    ycoordinate(6, 6) = 1
    xcoordinate(6, 7) = 2
    ycoordinate(6, 7) = 2
    xcoordinate(6, 8) = 2
    ycoordinate(6, 8) = 3
    xcoordinate(6, 9) = 3
    ycoordinate(6, 9) = 0
End Sub
```

```
Private Sub cmbChoice_SelectedIndexChanged(sender As Object, e As EventArgs) Handles
cmbChoice.SelectedIndexChanged
'combo box selection
Dim n As Integer
n = cmbChoice.SelectedIndex
If n = 0 Then
    lineshape = False
End If
If n = 1 Then
    lineshape = True
    prim.pshapenumber = 1
    prim.pshapecomponent = 4
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 1
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 2
    prim.pycoordinate(3) = 0
    prim.pxcoordinate(4) = 3
    prim.pycoordinate(4) = 0
End If
If n = 2 Then
    lineshape = True
    prim.pshapenumber = 2
    prim.pshapecomponent = 3
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 0
    prim.pycoordinate(3) = 2
End If
If n = 3 Then
    lineshape = True
    prim.pshapenumber = 3
    prim.pshapecomponent = 5
    prim.pxcoordinate(1) = 1
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 1
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 1
    prim.pxcoordinate(4) = 2
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 2
End If
If n = 4 Then
    lineshape = True
    prim.pshapenumber = 4
    prim.pshapecomponent = 6
    prim.pxcoordinate(1) = 1
    prim.pycoordinate(1) = 0
    prim.pxcoordinate(2) = 2
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 3
    prim.pycoordinate(3) = 0
    prim.pxcoordinate(4) = 0
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 1
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
End If
```

```
If n = 5 Then
    lineshape = True
    prim.pshapenumber = 5
    prim.pshapecomponent = 5
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 2
    prim.pxcoordinate(2) = 1
    prim.pycoordinate(2) = 0
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 2
    prim.pxcoordinate(4) = 2
    prim.pycoordinate(4) = 1
    prim.pxcoordinate(5) = 2
    prim.pycoordinate(5) = 2
End If
If n = 6 Then
    lineshape = True
    prim.pshapenumber = 6
    prim.pshapecomponent = 15
    prim.pxcoordinate(1) = 0
    prim.pycoordinate(1) = 1
    prim.pxcoordinate(2) = 0
    prim.pycoordinate(2) = 2
    prim.pxcoordinate(3) = 1
    prim.pycoordinate(3) = 1
    prim.pxcoordinate(4) = 1
    prim.pycoordinate(4) = 2
    prim.pxcoordinate(5) = 1
    prim.pycoordinate(5) = 3
    prim.pxcoordinate(6) = 2
    prim.pycoordinate(6) = 1
    prim.pxcoordinate(7) = 2
    prim.pycoordinate(7) = 2
    prim.pxcoordinate(8) = 2
    prim.pycoordinate(8) = 3
    prim.pxcoordinate(9) = 3
    prim.pycoordinate(9) = 0
End If
End Sub
Public Sub updatedisplay(dispa As Integer, dispb As Integer)
    'update display
    For x = (dispa * 10) To (dispa * 10 + 9)
        For y = (dispb * 10) To (dispb * 10 + 9)
            If pond(dispa, dispb) = "x" Then
                outputpic.SetPixel(x, y, Color.White)
            Else
                outputpic.SetPixel(x, y, Color.Blue)
            End If
        Next
    Next
End Sub
End Class
```