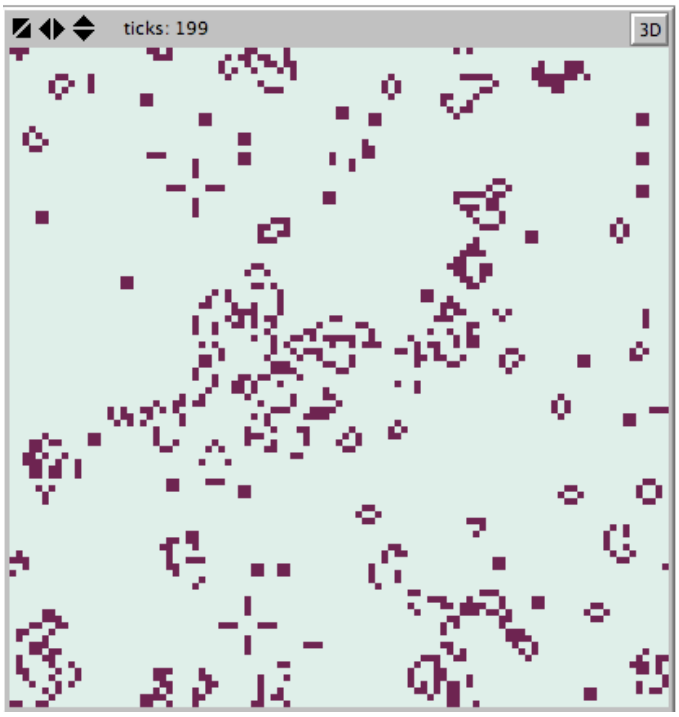
Introduction to Agent Based Modelling

NetLogo Guide 1

Patches - Cells



John Hayward
Church Growth Modelling
www.churchmodel.org.uk

Church Growth Modelling

Agents

An **agent** is an individual entity, or being, that can follow instructions and modify its behaviour in response to other agents and the environment.

There are four types of agents:

Patches, also called cells. These are static. They stay in the same relationship to those around them. They form a spatial grid sometimes called the world.

Turtles. These agents are dynamic, they can move around the world, i.e. over the spatial grid defined by the patches.

> Both Turtles and patches are "**alive**", i.e. they are active. The only major difference between them is that patches do not move. Agents have **variables**, or properties, that model their activities. E.g. if an agent is a person they could have an opinion either for a cause or against it.

Links are agents that connect two turtles

Observer. A bit like a global Turtle that looks over the whole world.

Agent Based Modelling

Agent based modelling investigates the mass behaviour of collections of individuals (agents) interacting with each other, based on the rules of each agent. Sometimes the agents are living things, animals, people, sometimes they are more abstract.

NetLogo

There are many programs to simulate agent based models. NetLogo is very flexible and free, available from

http://ccl.northwestern.edu/netlogo/

It is possible to use existing models without having to build your own, it has a large library. To build models requires learning a language which is a cross between programming and scripting. I.e. it has many built in features.

1 Basics

1.1 Patches

Patches, also called cells are the agents that are fixed to a grid. Their coordinates are fixed by integers on a two dimensional square. In order to understand patches you need to load up a library model.

Load NetLogo, go to the **Models Library** on the file menu.

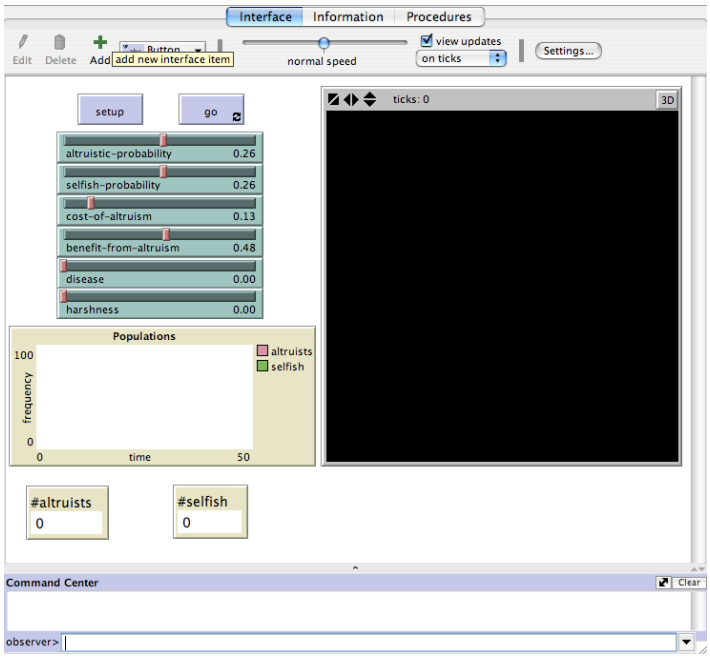
Open the **Social Science** Library and load "**Altruism**" . It is about population genetics and the selection of either a selfish gene or an altruistic one.

What you see is the interface and it should look something like:

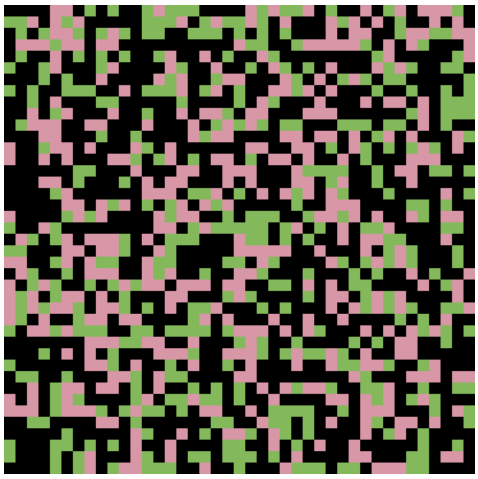
(This is the Mac version – your PC version may look slightly different!)

There is no need to understand the model at this stage

The interface is where you run the model. The back area is the "world". The grid the agents (patches in this case) are defined on.

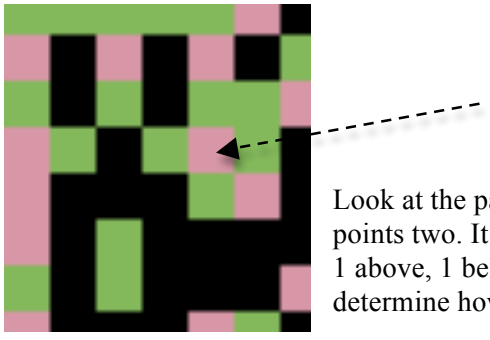


Click the button **set up** and you should see a variation on:



There are two different coloured patches. In my version pink are the altruistic agent and green the selfish one.

Each patch has neighbours, those that touch it.



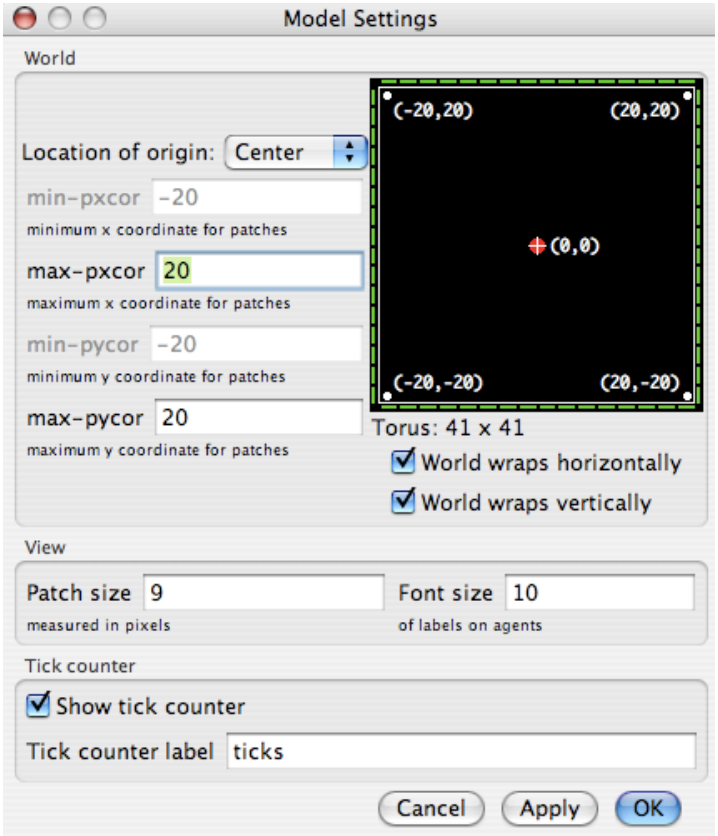
Look at the patch the arrow points two. It has 4 neighbours, 1 above, 1 below, 1 left and 1 right. The values of variables in here determine how the centre patch changes.

Often the diagonals are included and there are 8 neighbours. In this case our target patch has 5 green and one pink neighbours. The other 2 are blank.

1.2 Grid

Although Agent based models usually have a 2-d grid they do not have to. However it is much easier to picture if we keep to 2-d!

The size of the grid is tailored to each program by the writer. **Click** the button **settings.** You should see the details of the grid:



The total number of patches is determined by the settings min-pxcor, max-pxcor, min-pycor, and max-pycor. You can also choose where the centre of the world is.

Also the world can be "wrapped". I.e the left hand edge is identified with the right hand edge, and the top identified with the bottom. This is the normal practice. The boxes about wrapping horizontally and vertically are selected. It avoids problems of patches at edges have less neighbours than the norm.

Each patch has integer coordinates on this world. Thus there is a patch at (0,0), at (2,3) and at (-4,14). However there is no 21,20 as it is off limits. The patch coordinates are pxcor and pycor.

1.3 Running the Model

Go back to the interface and **click go**. The model runs. You may find after a while each agent becomes a selfish gene. This is a result of the model chosen for the way a patch interacts with its neighbours. The models computes the new value of each patch and updates the world

To stop it press go again.

There are controls that can change the behaviour of the model.

1.4 Setting Up The Model

If you want to be intimidated you can **click** the tab marked **Procedures**. This is the language that sets up the model.

This lab will teach the basics of setting up a model from scratch. Although some commands will be familiar from programming, this is not really a programming language as most of the important features are built in.

There is important features to watch out for are:

Variables
Agents have variables that represent its properties or atributes

Procedures
These are fragments of code that do things. The two important ones are **set up** and **go**. The ones you have seen. They begin with **to** and **end**.

You are free to define other procedures as you see fit. You would normally use them in **go** or **set up**.

Ask
This "asks" agents to do things

Set
Assigns a variable to a value

if ifelse
The one option and two option selection, works similar to programming

You will learn these though setting up the model called Game of Life

2 Conway's Game of Life – Constructing a Model

2.0 Describe the Model

Before you set up your model in NetLogo you need to be clear what the model is!

This game is a special type of agent based model called a **cellular automata**. We will call the cells patches.

Model of the Patch

This is the "data" model of the patch

Each patch has one attribute with one of two values, it is alive or dead.

Each patch has 8 neighbours, horizontal, vertical and diagonal

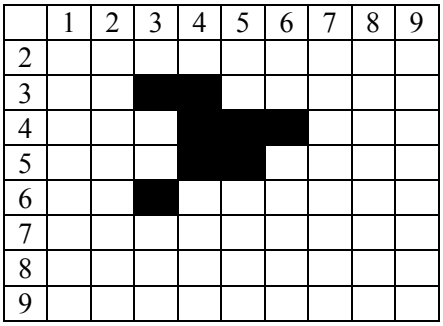
Dynamic Model

This is the model of how a patch changes over time.

Live patch becomes dead if either	it has 1 alive neighbour or less (dies of loneliness) It has 4 or more alive neighbours (dies of overcrowding)
Dead patch becomes alive if	it has 3 alive neighbours

Other patches stay unchanged.

Thus an alive patch with 2 or 3 neighbours survives to the next round.



Alive patches:

(3,3) 2 alive neighbours so lives

(4,3) 3 alive neighbours so lives

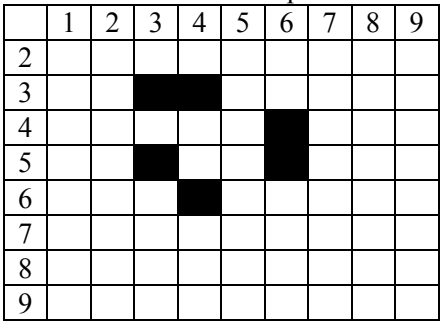
(4,4) 4 alive neighbours so dies

Dead patches

(2,3) 1 alive neighbour so stays dead

(4,6) 3 alive neighbours so comes to life

Thus after one time step this becomes



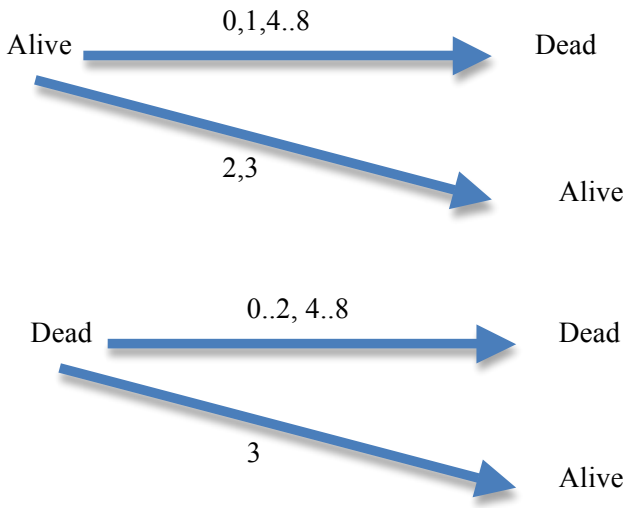
The decision to change is made on the old state and the new state is only constructed once finished.

The Game of Life does not model anything in particular. Its importance is to show how a simple agent based process works.

However it is of mathematical interest as it generates patterns that persist, patterns that oscillate, patterns that move and patterns that generate other patterns.

The rules of the model are simple, the behaviour is very complex.

The rules can be summarised by state transitions:



NB These roles are the heart THE MODEL!

The properties of the Cells are also part of the model

Thus the model will consist of :

- Data Model
- The Rules

Now to set up the model

2.1 Set the variables of the agent

Defining the patch or agent must come first.

Start a **new** file from the file menu.

Go to the procedures tab, which should give an empty box.

In the following
commands that are in NetLogo are in normal characters
names we invent are in *italics*

However do not type italics!! This is just so you realise what is a NetLogo command.

The Data Model:

A patch has two variables
1. One to indicate whether it is alive or dead, call it *living?*
2. One to count how many alive neighbours it has, call it *live-neighbours*

Living? is a boolean, i.e. it is either true or false. The question mark flags that.

live-neighbours is an integer.

Thus type in:

```
patches-own
[
  living?
  live-neighbours
]
```

The command patches-own is NetLogo saying what variables a patch has. The variables are between square brackets.

Save your program.

2.2 Think Out Procedures

It is unlikely you can work out either set up or go without some procedures. However this section often has to be come back to as you realise you have missed things.

Think what happens, live cells become dead, dead cells become alive. Thus these two are needed as procedures.

Type in the procedure for a cell becoming alive:

```
to cell-birth
  set living? true
  set pcolor magenta
end
```

The "to" and "end" bracket the procedure and we have given it the name *cell-birth.*
living? Is made true – that means it is alive
The colour of the cell is changed magenta.

set assigns the variable.
pcolor is the NetLogo variable for the colour of the patch (USA spelling)

There was no reason why we could not use our own colours, but fgcolor is the defaults foregrounds colour so that is good enough.

The procedure for a cell die is fairly easy to guess. **Type in**:

```
to cell-death
  set living? false
  set pcolor white
end
```

Now we can tell which one is alive and which one is dead.

2.3 Set Up The Blank Screen

Initially the screen should be blank. It makes sense to have all cells dead. **Type:**

```
to setup-blank
  clear-all
  ask patches [ cell-death ]
end
```

"ask" tells every patch to run the command *cell-death*. Thus every cell is killed. We know every cell runs the command as "patches" is plural!

"clear-all" just re-sets all variables, including the clock to the original values. Always do this. It means you can set up again and again after you run a simulation.

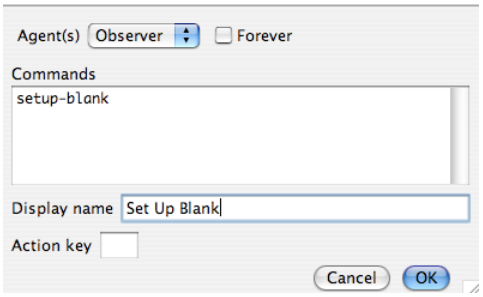
Note that *setup-blank* is still a procedure, but it is specifically used to start the simulation off.

2.4 Start interface

There is not enough to run a simulation but it is worth seeing how far we have got. **Click** interface tab.

To add a button for setting up the blank screen.

1. Ensure that "Button" is showing on the drop down menu.
2. **Click Add** and **click** on the area to the left of the black screen in the place you want the button.
3. **Type** in the name set-up-blank under Commands and **type** your own name under Display Name. Something like:



4. **Click OK**.

If you now click the button the screen is made full of dead cells. You know they are dead because of the colour.

All items on the interface can be dragged around.

Remember to keep saving your file!

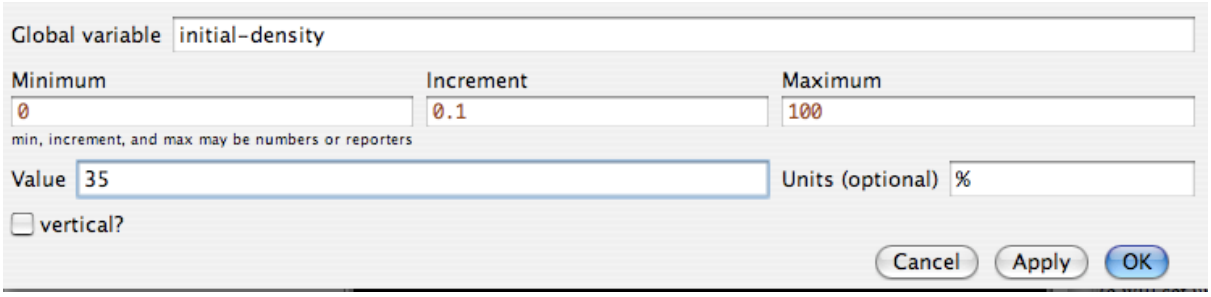
2.5 Set Up Initial State

To start the simulation off some alive patches are needed. We will set up a random arrangement of patches, using some simple probability. Go back to the procedures tab and type:

```
to setup-random
  clear-all
  ask patches
    [ ifelse random-float 100.0 < initial-density
      [ cell-birth ]
      [ cell-death ]
    ]
end
```

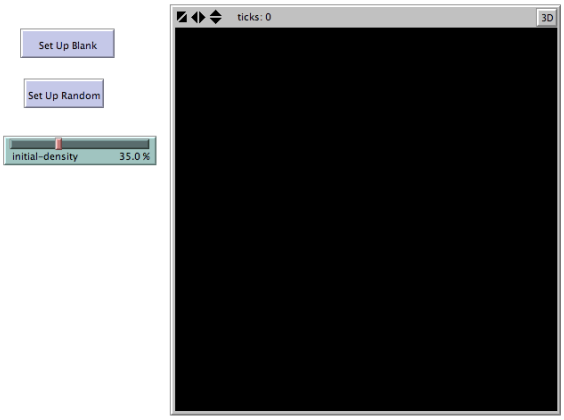
1. A new variable *initial-density* has been brought in.
2. ifelse is a statement with two options, the first is if the if is true, i.e. cell-birth, the second is if the if is false, cell-death.
3. random-float 100.0 chooses a number between 0 and 100.
4. The user will choose initial-density to determine roughly how many alive patches there are, so an initial density of 50 should ensure about equal coverage of alive and dead
5. Initial density will be set on the interface.
6. Again ask ensures that each patch caries out the instruction. Ask is a bit like a for loop that runs through all patches.

Go back to the Interface. **Select** the **slider** from the drop down menu, **click add** and click on the are for controls. The dialogue appears.

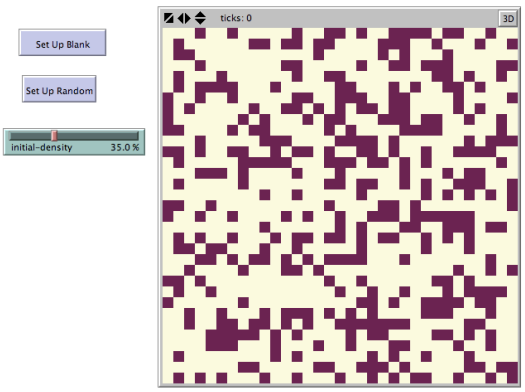


Initial density a global variable. It is given limits of 0 and 100, and an increment of 0.1. The 35 is the initial value it is given when the program is first run

Also set a button for the Set up Random. Your interface should now look like:



Click Set Up Blank then click Set Up Random, it should now look something like:



Your patches are ready for action!

2.6 Define Main Procedure

The rules to cause the patches to change is now needed. It is normal to call the main procedure "go". The main procedure will do the following

Update the number of alive neighbours each patch has:

1. Ask all the patches to count how many alive neighbours they have. This is
   count neighbors with [*living*?]
   Note the NetLogo command neighbors is the USA spelling. It questions the 8 patches surrounding the patch under question.
2. The number is put in the variable live-neighbours.
   set *live-neighbours* count neighbors with [living?]
   Thus the variable *live-neighbours* is set to the number of alive neighbours for each patch. Remember each patch will have a different value to this variable.

Decide whether there are the right number of alive neighbours to kill off an alive patch, or bring to life a dead one

3. Ask each patch about the number of alive neighbours it has
4. For alive patches 2 and 3 neighbours live and the rest die
5. For dead patches 3 neighbours live and the rest die

Type in:

```
to go
  ask patches
    [ set live-neighbours count neighbors with [living?] ]
ask patches
   [ ifelse living?
        [ ifelse live-neighbours = 2 or live-neighbours = 3
            [cell-birth]
            [cell-death]
        ]
        [ ifelse live-neighbours = 3
             [cell-birth]
             [cell-death]
        ]
   ]
  tick
end
```

These rules are the heart of the model, change the rules and you change the model.

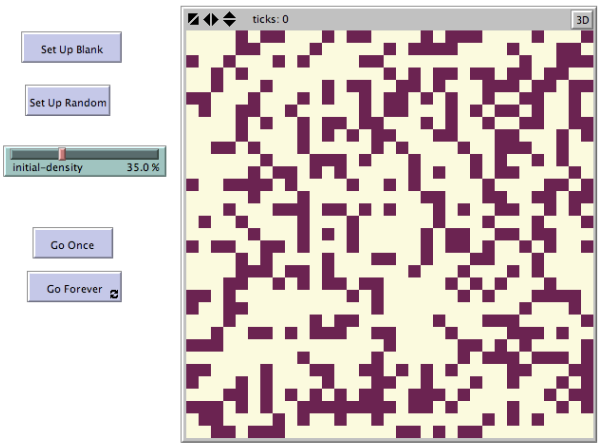
The command tick updates the clock.

Go to the interface and set up TWO buttons.

1. One called Go Once which uses the procedure go
2. One called Go Forever which uses the procedure go and where the forever check box is checked. This will be the main run procedure

2.7 Run The Simulation

Your interface should look like



One last thing – under view updates change continuous to on ticks. It just runs a bit better!

1. Click Set Up Blank
2. Click Set Up Random
3. Click Go Once. You will see the change after one round. Click again that is the second round.
4. Click Go Forever. The program runs.

Eventually it will stop, when it does you can stop the simulation by clicking go-forever again.

Try Some Experiments

You get a different result each time. This is due to the randomness in the set up.

Try and spot the following

1. Are there times when all the patches die?
2. If you lower the density will the cells all die?
3. What happens if the cell density is very high?
4. What patterns are stable?
5. What patterns move?

3 Getting More Out of the Model

3.1 Simple Output

Monitor
As there are occasions where some cells are left it would be nice to know how many are left.

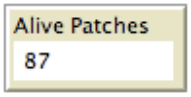
Go to the interface and choose **monitor**. Click add and **place** the monitor on the interface. In the reporter **type**:

```
count patches with [living?]
```

It does exactly what it says. Counts patches but just filters the one that are living.

Give it the title Alive Patches, and choose 0 decimal places.

Run the model and you will see the Alive Patches at each time. Then you can see how many are left. It should look like:



Run a few simulations and get a feel for how many patches survive for different densities.

Plot
A plot of how many alive cells can be done as the mode runs. This will take a little more work. There needs to be a procedure to set up the plot and one to do it.

For setting up the plot add in the procedures tab:

```
to setup-plot
 set-current-plot "Life"
end
```

It just names the plot “Life”.

To do the plot type in the procedure:

```
to do-plot
 set-current-plot "Life"
 set-current-plot-pen "alive"
 plot count patches with [living?]
end
```

This names the plot “alive” on the plot “Life”. It plots the result of counting how many patches are alive.

These two procedures need to be added to the random setup (bold)

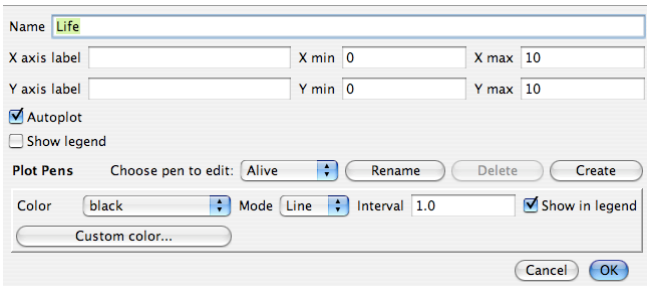
```
to setup-random
  clear-all
  ask patches
    [ ifelse random-float 100.0 < initial-density
      [ cell-birth ]
      [ cell-death ]
    ]
    setup-plot
    do-plot
end
```

do-plot is then put after the "tick" in "go". Thus each point is plotted after the state is updated:

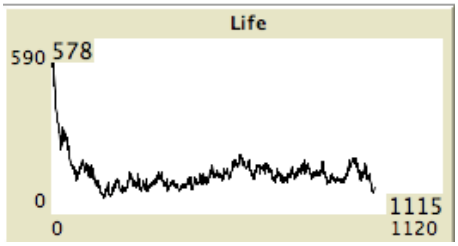
```
to go
……..
  tick
  do-plot
end
```

Go to the interface, select **plot** from the drop down menu, then click **add** and place a plot on the interface

1. Give the plot the name Life – ties it with your plotting procedure
2. Rename the pen to Alive – ties it with your plotting procedure
3. Ensure autoplot is checked



Run the model and you should see something like:



3.2 Choosing Your Own Starting State

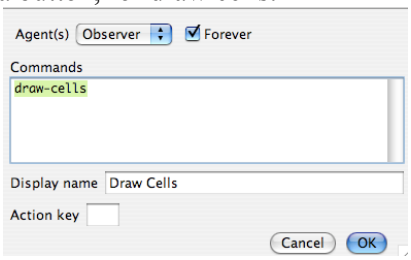
A initial random state is fine, but it would be nice to choose our own. This is yet another procedure, one that responds to the mouse. Type in

```
to draw-cells
  let erasing? [living?] of patch mouse-xcor mouse-ycor
  while [mouse-down?]
    [    ask patch mouse-xcor mouse-ycor
         [ ifelse erasing?
              [ cell-death ]
              [ cell-birth ]
         ]
        display
  ]
end
```

This is a bit harder to follow. A local variable *erasing?* Is set to be true if the patch at mouse-xcor mouse-ycor is alive. This the pacth the mouse is hovering over.

When the mouse is pressed if it is *erasing?*, that is alive, then it made dead, and vice verse.

Now on the interface layer set up a button, for draw cells:

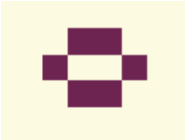


Make sure it is “Forever”.

To run the model click Set Up Blank first, then click Draw Cells:

Click the mouse on the World and do a line of four patches:

Keep clicking “Go Once” and it quickly stabilises to fixed pattern.



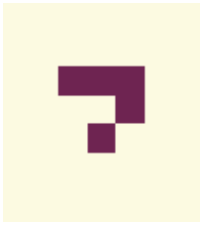
This is one of the game of life fixed patterns

3.3 Some Game of Life Patterns

1. Try the 4 cells in a block. This is stable
2. Try a line of 5 this becomes an oscillating pattern
3. Try a line of 6. It eventually disappears
4. Try a line of 10. This oscillates and the number of cells jumps between different values

Glider

There are patterns that "move" try:



Click Go Forever and the pattern will move upward and to the right.

Search for Game of Life on the internet and you will see many other patterns.

Try drawing some random patterns on the world.

3.4 More Patches

The size of the world can be changed. At the Interface layer go to Settings.

Change the maximum corrodinates to 40 in each direction max-pxcor and max-pycor

Change the patch size to 10

Make blank, set up random and run. There are now many more patches

3.5 Change the Topology of the World

Topology refers to shape. In this case we can switch off the wrapping and give the world an edge. In settings unclick both the wrapping boxes.

If you randomise and run now you will find some stable patterns will emerge near the edges as they can go no further.

5 Voting Model

5.1 Describe the Model

Although Game of Life is not a proper model of anything it is good fun and shows how simple rules generate complexity when there are many such agents. However cellular automata, i.e. an agent model based on fixed patches is used for serious models. One area is that of opinion dynamics. The voting model is an example of opinion dynamics.

Each patch represents a person with an opinion as to how to vote. There are two voting options called 0 and 1.

The rule is if a patch is surrounded by 5 or more patches of a different opinion, then the patch will change its opinion to match the majority. The person changes their opinion.

To achieve this a patch will count up the votes around them. Thus if they are 0 and the total is 5 or more they become a 1.

However if they are a 1 and the total around them is 3 or less they become a 0.

Start a new model, and

5.2 Data Model

There will be two variables, the vote or opinion, with values 0 or 1. Type

```
patches-own
[
  vote   ;; my vote (0 or 1)
  total  ;; sum of votes around me
]
```

The semi colon is a comment. It does not do anything but can remind you of what you did.

5.3 Helpful Procedure

Because the colour of a patch will often change then we wil have a procedure to colour a patch

```
; procedure to set the colours
to recolor-patch
  ifelse vote = 0
    [ set pcolor green ]
    [ set pcolor blue ]
end
```

Green are 0 and blue are 1.

5.4 Screen Set Up

To set up the screen choose a random number, either 0 or 1. This is done with random 2. The pacth colours are set from this number. Type

```
to setup
  clear-all
  ask patches
    [   set vote random 2
        recolor-patch
    ]
end
```

Remember ask patches ensures this is done for each patch.

5.5 Main Procedure

As before the first step is to update the total for each patch. Then use that information to check whether the opinion changes.

```
to go
; update the opinion of a patches neighbours
ask patches
    [ set total (sum [vote] of neighbors) ]
; check whether opinion changes
ask patches
    [   if total > 4 [ set vote 1 ]
        if total < 4 [ set vote 0 ]
         recolor-patch
    ]
  tick
end
```

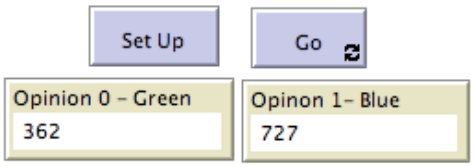
5 or more means 5 or more neighbours are 1, thus patch becomes 1 (blue)
3 or less means 5 or more neighbours are 0, thus patch becomes 0 (green)

5.6 Interface
Set up two buttons one for set up and 1 for go (click forever). Set up two monitors, one to count the number who vote 0 (green), use in the monitor box:

count patches with [vote = 0]

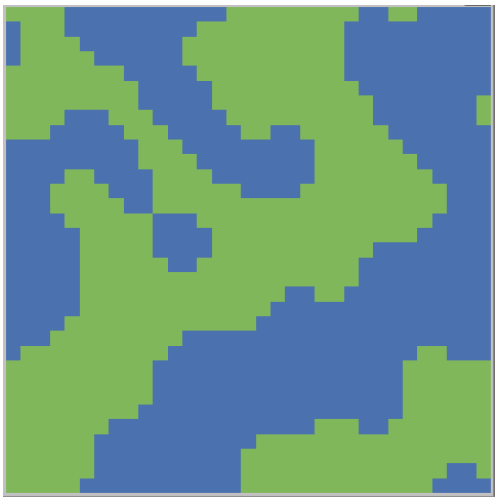
and another monitor to count those who vote 1 (blue).

You very basic interface should look like:



5.6 Simulations

Press set up and press go. You will find that like votes lump together, something like



Often one will end a little bigger than the other. It is unlikely one opinion disappears altogether.

Does the number of cells make a difference?

Go to settings change the maximum coordinates to 70 and the patch size to 4. The model takes longer to run but similar results are produced.

Models of Opinion Dynamics of this sort are huge area of social research. They are similar to the behaviour of magnetic solids and are sometimes called the Ising model.

6 Exercises

1. Change the model of section 5 so that people who have neighbours which are entirely like themselves will change opinion, just to be different. So total =8 becomes a 0, and total = 0 becomes a 1.
2. Change the model of question 1 so that those with total 8 change randomly, so some change to be the opposite of their neighbours some don't.
3. Change the model of section 5 so that those with an equal number of different opinion neighbours change randomly, i.e. total = 4.
4. Change the model of question 2 so that those with neighbours where there is only a majority of 1 change randomly, i.e. total = 3,4,5.
5. Change the model of section 5 so that people only change with a probability of a half.
6. Change the model of section 5 so that there are now 3 opinions, with values 0, 1, 2, where 2 is the middle ground. Examine different boundaries where the opinion changes. Are there any models where all 3 opinions survive? Remember Total could now go up to 16. You might like to give the boundaries names and choose the values with a slider