Two practical challenges to introduce StarLogo TNG and common techniques associated with model building. Two further kinaesthetic activities to illustrate decentralised systems and the primacy of local communication.

Preparation required:
StarLogoTNG installed on all computers. Hungry Turtles and African Plains programs available for all. Blindfolds (scarves) and Post-It notes for Blind Mice activities

Dynamic Agent Based Models

There are lots of tools for building deterministic computer models, but they all rely on the same approach – articulating the rules that determine the aggregate behaviour of the model. This presents barriers to students developing anything beyond a trivial centralised, deterministic model. In this section we'll introduce a tool for modelling decentralised systems - that is, systems where there is no central control.

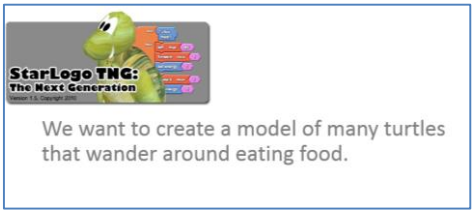
Such systems are much easier to model because complex behaviour can be based on very simple rules. What is more, they are very commonplace, particularly in the natural world. Think, for example, of how an epidemic spreads. Nobody organises it – though many agencies try to prevent it. A lot of animal behaviour can't be explained by reference to a 'leader' or co-ordinator either. Think of flocks of birds or schools of fish. Often group behaviour such as this is so remarkable it is difficult to conceive that there is no central agency co-ordinating it. Patterns of erosion might be another example, or the remarkable shapes of desert dunes. Simple eco-systems like the marine model which introduced the Unit, or pond life. The dynamics of predator – prey relations can be explored and so on… the list is endless.

Such behaviour is not just decentralised, but dynamic too. Dynamic in that the outcome cannot be predicted in advance. In contrast to deterministic models, where the same parameters will always provide the same outcome, these are stochastic models – models that include an element of randomness. As such they provide very stimulating environments for children to explore, relating to the real world and encompassing a degree of unpredictability.

What's more, we can build such models with very limited experience of maths or programming. Most students at KS3 will have some familiarity with Scratch and the tool we'll use, StarLogoTNG, bears a remarkable similarity.

Hungry Turtles

StarLogoTNG was developed by Mitch Resnick, who led the team who developed Scratch. It has evolved over many years. There is a further evolution – StarLogo Nova (goo.gl/5U3v2h). Like Scratch2, Nova is an online version that requires no installation. It offers other improvements, but is still in its infancy so our focus remains on StarLogoTNG. When Nova is developed further, it should be easy to carry the ideas over to the new environment. StarLogoTNG can be downloaded from the MIT website: goo.gl/67z9Rv. There are versions for Windows, Mac and Linux. We start with a simple exercise to get familiar with the environment.



We create a world of turtles that wander around eating. Start by decomposing the challenge. Breaking it into smaller parts makes it easier to handle. A series of clicks outline 3 stages. Tackle this as a group walkthrough. Navigating new software as a group can be difficult so encourage everyone to keep an eye on their neighbour. Sample files, showing the end of each of the three stages are available. Pause at the end of a stage. If anyone is lost, use the sample file – this allows them to pick up from a common point.

Stage 1: Launch StarLogo. Two windows open. Focus for the moment on the right hand window. This is known as SpaceLand – it is the environment the model runs in. By default you have an aerial view of the world. Use the arrow keys and +/- to move the camera (the view) around. Also experiment with the view buttons at the top. These allow you to view the world from different perspectives. Agent Eye sees the world as one of the 'agents' sees it. The Agent View positions the camera behind the 'agent' so you can observe what it is doing. The small inset shows an overhead view – Swap Views will make this the main view and put the previous selection in the inset.

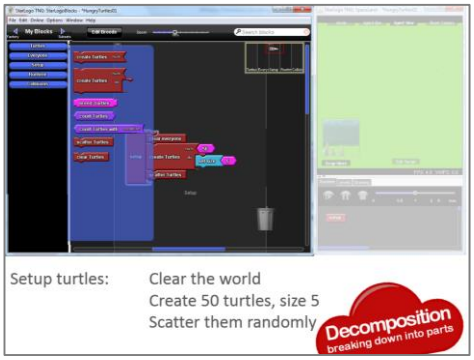
Notice there are facilities to edit the terrain. This reveals it is a flat grid of squares, known as patches. You can create a more varied terrain, but discourage this in the early stages. The key to a good model is simplicity. We look at varying the terrain later.

The left hand window is where code is assembled. There are several areas within it for code, rather like in Scratch, where each sprite has its own code area. A default world has one 'breed' of agents – turtles, so there is an area for coding turtle behaviour. The other areas are shown in the thumbnail inset highlighted. Understanding where to place code is an important first step.

Let's consider how to populate our world with lots of turtles. The code we assemble will need to go in the Setup area. Code in this area creates the world we wish to start from. Use the scroll bar to navigate there. The stages required are outlined on the slides. It's always a good idea to clear the world first. The main code blocks required can be found in the Setup and Run drawer. Here you'll find Setup and Clear to assemble the code shown, but where is the Create Turtle command?

Notice the top left controls. The default drawers are part of the general Factory of commands – think of them like keywords in a traditional language.

The commands to create Turtles are specific to that 'breed' or object. You can create as many breeds as you like – they are yours to define. Although a model is created with a Turtle object as a default, the commands associated with it are in a different set of drawers; My Blocks. The Turtle drawer contains a set of default actions that are the same for any breed you create – namely the ability to create, delete and scatter them. There are also commands that allow us to count them – note how they are colour co-ordinated to the respective drawers in the Factory.



Once you have defined your Setup procedure, you will notice a setup button in the SpaceLand. Click it to run the procedure. Now would be a good time to save the project.

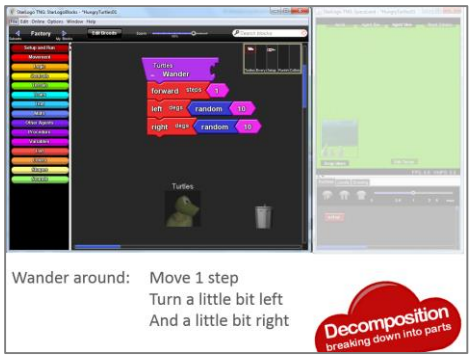
Stage 2: To recap, we initially identified 3 stages to developing our model. The first is now complete, so let's consider the second stage; getting our turtles moving. Again we can decompose the action further.

We want to code a 'wiggle walk' moving forward each time but turning a random amount before taking the next step. Having identified the actions, we need to decide where to put the code. Look at the options. Wandering will clearly take place at Runtime, but the actions are specific to Turtles.

The concept of a procedure is key here. We can define the procedure in the Turtle space, then call it when the model runs. With the Factory visible, select the Procedure drawer and drag a Procedure block into the Turtle space.

Notice that as you drag the block to the Turtle space, StarLogo assigns the procedure to the breed. Give the procedure a sensible name, such as Wander.

We can attach the commands for the wiggle walk below the name. The step forward is straightforward, but what about the turn? A 'wiggle walk' procedure is a common technique used in many models. Once defined we can call it from the Runtime space.



Navigate to the Runtime space. We want the Turtles to continue wandering. Look at the code options in the Setup and Run drawer - which should we use? If you hover the mouse over each block it gives a summary of its actions.

We want a 'forever' block. As you drag it to the Runtime space, change its name to Run, Start or something similar. This will be the name displayed on the button that will activate the code in SpaceLand.

When the Run button is clicked, we want it to call the Turtles Wander procedure, but where will we find it? Having defined the procedure for the Turtle breed, you'll find a Wander block has been added to the Turtle drawer in 'My Blocks'.

Encapsulating a sequence of code in a named procedure means we can hide the detail from the main Runtime code. Defining and calling procedures are a good example of how decomposition and abstraction can work hand in hand in programs. We decompose a problem into its smallest parts and define them as procedures. The procedure calls abstract away the detail involved, leaving the main runtime code easier to comprehend.

Now you have defined your Run code, a new button has appeared in SpaceLand so you can test it. Once working, make sure you have saved it. It's useful to save with a separate version number, so if you make a mistake as you build a model you can always revert back to an earlier version. We've now completed two stages of our model.

Stage 3: Let's turn our attention to the final stage – eating. The steps involved are outlined on the slide. Our model is a simplification. All we want to do in this familiarisation is indicate how eating might be represented. We can do this by changing the colour of a patch.

As before our first decision is where to place this code sequence? As it applies to Turtles we should navigate back to the Turtle space. Having decided where to put it, what block should we select first? A procedure block, so we can define the sequence and give it a name.

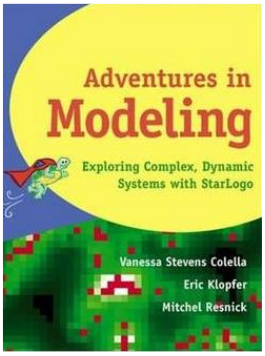
On the next slide a procedure block has been dragged in and given the name Eat. The code involves a selection construct – an IF statement. Where might we find these? In the logic drawer. Armed with this information challenge attendees to finish the model themselves. The solution is given on the following slide (shown).

With the procedure defined, all that remains is to call it. Notice how, by abstracting away the detail, the behaviour of the model is very clear. Once added to the Runtime code, we can test the model and save it.

We started by noting the similarity with Scratch, but even a simple introduction like this should make it obvious that it is more complex. StarLogo has a higher floor of entry than Scratch – it is not a beginner's environment. Crucially, some familiarity with the notion of procedures is a prerequisite. Some years ago Jeremy Scott produced three excellent introductions to block based coding funded by The Royal Society in Scotland. They are available, along with supporting videos, free from goo.gl/VolqZb. The booklets provide progression using block based languages, starting with Scratch, using BYOB (now SNAP!) to introduce procedures before moving on to App Inventor. StarLogo projects sit in a similar place to App Inventor. Some familiarity with the concept of procedures is an essential prerequisite for pupil success.

A Round Of Applause

There are lots of software tools for building computer models, but they all rely on the same centralised approach – articulating the rules that determine the aggregate behaviour of the model. As we've already noted, this presents difficulties for students to develop anything beyond a trivial model. The simple example we just built demonstrates the potential of a different, decentralised approach. The code can be very simple, but the overall impact, with many agents executing simple commands can be impressive. A simple kinaesthetic activity can help get over the idea of decentralised behaviour.



It's taken from a book, 'Adventures in Modeling', written in 2001. Ask the group to clap and adjust their clapping so everyone is clapping in unison. There is no central direction, yet it is remarkable how quickly people converge on the same beat. Ask them to consider what cues they used to converge – there is no conductor.

Repeat the exercise with eyes closed. This removes any visual cues. Does it take longer? Auditory cues can be removed by asking the group to synchronise raising and lowering their arms. Was this easier or harder? Did it take longer? A final extension activity can remove both auditory and visual cues. Stand as small groups in a circle holding hands. With eyes closed and in silence can you synchronise squeezing hands?

All these activities are examples of decentralised behaviour. No-one co-ordinates the activities yet they result in synchronised behaviour. The greater the communication, the easier it is to achieve, but as we shall see later, even with limited communication startling behaviour can be observed. Later we develop these ideas further.

On The African Plains

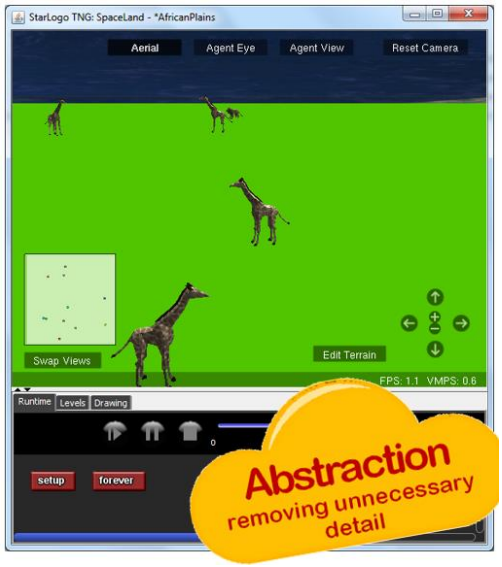
For our second activity we develop the idea of modelling an eco-system. This is a theme that runs through the day. The ideas for these activities came from Project GUTS: Growing Up Thinking Scientifically. Based at the Santa Fe Institute, they have provided a lot of material for exploring modelling in science. As well as material shared with Code.org, they have organised two 'moocs' to introduce the ideas of modelling Complex Adaptive Systems. One is based on StarLogo TNG, the other Nova. More details on their website: projectguts.org/.

Now that we have a basic understanding of how a model is created in StarLogo, we'll focus on exploring an existing model. This allows children to 'read' existing code before being tasked with 'writing' it. It makes a good paired activity. Children work together to analyse existing code and investigate how a model works. This can lead on to paired programming activities where they extend the existing model.

Open the Star Logo project African Plains, and run the Setup procedure. Don't run the model yet, but spend a moment observing what is in SpaceLand. Zoom in and look round – what do you see? We have several giraffes scattered around the terrain. If you look closely you'll see a lion too.

This is another example of abstraction – we might call it representational abstraction. The model represents the African Plain but it is stripped of any unnecessary detail.

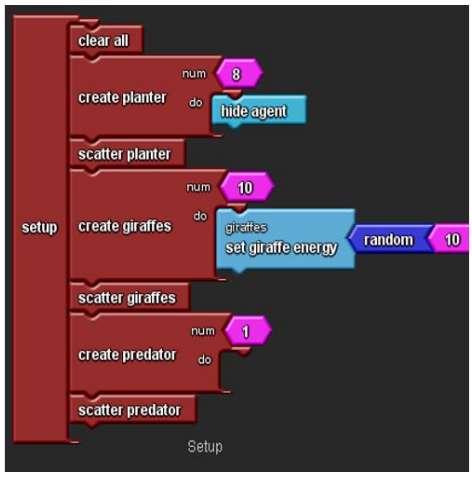
We want to model some basic behaviour of a predator and their prey, so our model includes two animals and little more. As it is modelled on a computer, using software, we can think of it as a computational abstraction.

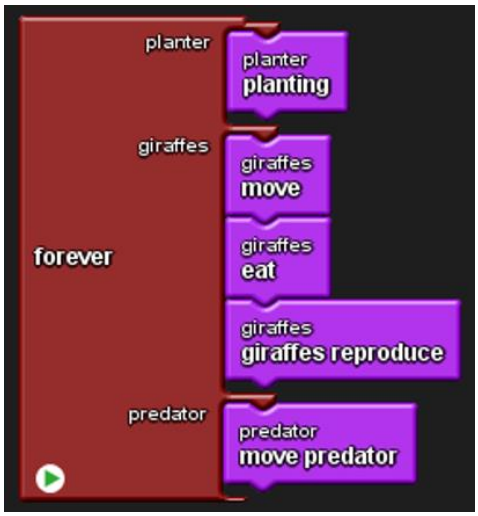


We don't need to get obsessed with the terminology. The more important point to make is that 'abstraction' occurs again and again in a variety of different forms. In all its forms it acts an aid to understanding – removing or hiding detail, so we can focus on what we want without getting bogged down in needless complexity.

Now run the model, by selecting the Forever button. What do you observe happening? There is a bit more to this model than meets the eye. The predator runs around – does it eat the giraffes? (Yes – when they touch, the giraffe disappears) What do we notice about the giraffes? (They also run around and new ones are born) What is happening to the terrain? (Some patches turn yellow, which indicates growth of plants and the giraffes eat them).

To understand what is going on we need to turn our attention to the code. Start with the Setup procedure. How many different 'breeds' are there in the model? Notice there is a third breed 'planter', as well as the predator and giraffes we have observed. What is the difference between a 'planter' and the other two breeds? It is hidden so we can't see it. What do we notice about the Giraffes? They have an energy variable. This will play a crucial role in their later behaviour. What value is it set to when each Giraffe is created?

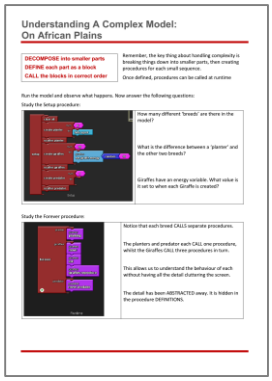




Let's now study the Forever procedure in the Runtime environment. Notice that each breed CALLS separate procedures. How many procedures do each breed call? The planters and predator each call one procedure, whilst the Giraffes call three procedures in turn. Without delving any further, can you summarise the behaviour of each breed? (The planter's plant, the predators move. So do the giraffes, but they also eat and reproduce)

We can understand the behaviour of each breed at this high level without having all the detail of how they do each thing cluttering the screen. The detail has been ABSTRACTED away. It is hidden in the procedure DEFINITIONS. Another example of abstraction – procedural abstraction allowing us to understand the overall behaviour of the model.

It's worth reiterating this with students. Decomposition and abstraction can be two sides of the same coin. You might want to get them chanting it; "When I say abstraction you say…" And "When I say decomposition you say…" It is the key to understanding complex code. We've completed the first page of the activity sheet. In pairs work through the rest of the exploration making notes. In 10 minutes or so gather the group together to check understanding. The explanations that follow accompany the slides displaying each question in turn. You may wish to hide them if they aren't required but explaining code aloud is a very good test of understanding. The activity sheet includes practical extensions at the end which can form the basis for an extended lesson, after the code has been discussed.



Collisions are a key concept in agent based modelling. We can specify the behaviour of any agent when it collides with another. Notice in My Blocks – Collisions drawer there is a block for every possible collision combination.

As you add new breeds to a model, new collision blocks are automatically added.

The move predator procedure should be clear. It's our 'wiggle walk' with a bit more wiggle! We've taken a technique from a previous project and 'generalised' it. We've applied it to a new context with different parameters to suit that context.

Throughout the projects we investigate, you'll find similar examples of general techniques. By the end, we hope to have a toolkit of these common techniques to apply in our own model.

The planting procedure is another more extreme wiggle walk. The conditional statement generates a random number between 0 and 10. It tests to see if it is 1. If it is, it stamps the patch it is on yellow. So there is a 10% chance that a patch will start 'growing'. We could alter this to slow down or increase growth.

Notice again, with the Giraffe's move procedure how we rely on the wiggle walk! The critical factor here is the giraffe's energy. This is set to a random number between 0-10 when the giraffe is created. It declines by .01 on each iteration of the forever loop. If it reaches zero the giraffe dies. You can experiment by altering the amount of energy a giraffe gets on creation as well as the amount it declines each step.

The eat procedure relies on a conditional statement. It tests the colour of the patch it is standing on. If it is yellow it does two things; it changes the patch to green and increases the giraffe's energy by 1. To offset the energy lost by walking, a giraffe has to eat every 100 steps. Again, we can experiment with the effect of altering the energy boost from each 'eat'.

The reproduction procedure is the most complicated. It is shown left. Study it carefully. Reading code is a quick way of assessing understanding and precision. Ask a child to 'read aloud' the procedure. Taking that as a starting point, one technique might be to encourage children to successively improve on the explanation. Multiple contributions allows time for those who struggle to comprehend the code to build their understanding. Finally, the teacher can model a precise answer.

Again, this procedure relies on a conditional construct. If the giraffe's energy is greater than or equal to 10, two things happen. First a Hatch procedure is invoked. Secondly the energy of the 'mother' is halved.

A common mistake when articulating code is to focus on the detail. The explanation above mentions a hatch procedure but omits any detail. However the detailed sequence of the Hatch procedure does need to be explained as a further, separate step.

A 'Hatch' is a built in block that creates a new instance of the breed. The new giraffe 'agent' has its energy set, just as those created at setup do. But rather than be scattered, the agent heads in a random direction a few steps away from the 'mother'. This is a common technique to 'spawn' new agents in models. Moving them away allows several to be born and not all be on top of each other. Finally, the new born announces its arrival!

Blind Mice

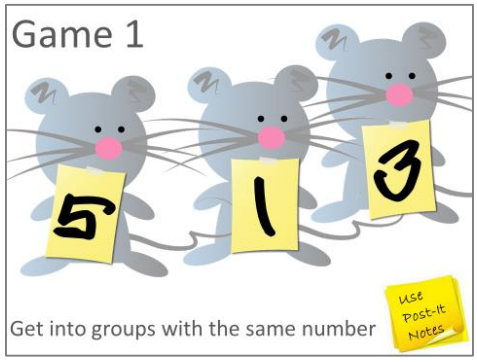
This is another kinaesthetic activity taken from 'Adventures in Modelling', though its origins lie in an interesting lecture paper written by Mitch Resnick and Uri Willensky. The paper is well worth reading for a source of other ideas about role playing activities. You can find it at: goo.gl/Jh44gr. A video explaining the activity, from Project GUTS is included in the resources.

In any multi-agent system, the agents communicate in some way. In 'round of applause', auditory cues played a key role in synchronising clapping. Even in non-human systems, agents communicate. When snooker balls collide, they communicate information about their velocity and direction.

In centralised systems communication is mediated by 'leaders'. Conductors direct orchestras for example. In decentralised systems, such global communication is minimised. Local communication is the norm - it limits information exchange to that between individual agents. The following activity is designed to explore the differences between global and local communication.

Blind mice is 3 distinct games. The goal of each game is the same but participants are encouraged to think about the different communication strategies involved. You will need a supply of Post-it notes and some scarves or other material to use as blindfolds. It is best conducted outside or in an open area without obstacles.

Game 1 involves each student picking a random number between 1 and 5 and writing it on a post-it note. If groups are under 20, try numbers between 1 and 4; under 12, between 1 and 3. They should not show others their numbers. Without any further prompting, ask them to organise themselves into groups with the same number e.g. all the ones together. Do not tell them how to do it!



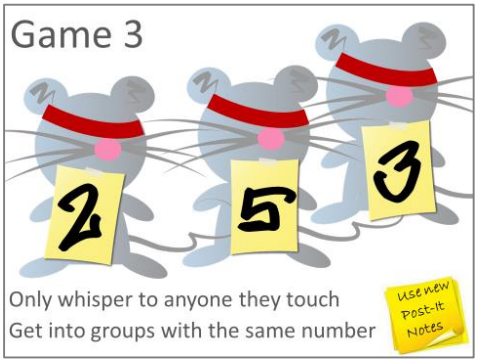
When done reflect on what strategies were used to move into the correct groups. Who acted as leaders? What did the leaders do? Did they shout out? Who acted as followers? What did followers do? Did they just move to the correct place? How were leaders chosen?

In this exercise participants usually employ centralized, global strategies. A small number organize the others.

Game 2 involves each student picking a different random number between 1 and 5 and writing it on another post-it note. They should not display their numbers publicly but can show them to their immediate neighbours. This time no talking is allowed. No arm waving, stomping of feet or other signals either! Without any further prompting, ask them to organise themselves into same number groups again. Do not tell them how to do it!

When done reflect on what strategies were used to move into the correct groups. In this exercise participant are forced to rely partly on local communication, but they can observe global behavior, such as groups forming and so can gravitate towards them. What strategies were adopted to find people with the same number? Which were most successful? How did they group together? Did only partial groups form in some cases?

Game 3 involves each student picking a different random number again on another post-it note. This time they are blindfolded so can't see any numbers. Some children may need to act as helpers to guide people around, so anyone uncomfortable with being blindfolded can be employed to 'watch over' the group. Blind mice can whisper their number to anyone they bump into. Without any further prompting, ask them to organise themselves into same number groups again. Do not tell them how to do it!

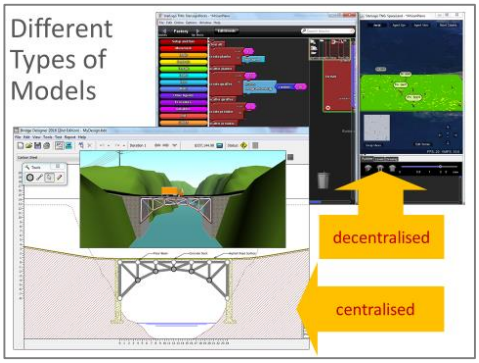


At some point stop the activity. Some groups, or partial groups may have formed. Before removing the blindfold reflect on the experience. Was it harder to find like numbers? Why? How did you feel when you found the same number? With blindfolds removed, again reflect on strategies to identify correct groups. Did the strategies differ from the first or second games? Which were successful? Did only partial groups form in some cases? Did any not find a like number?

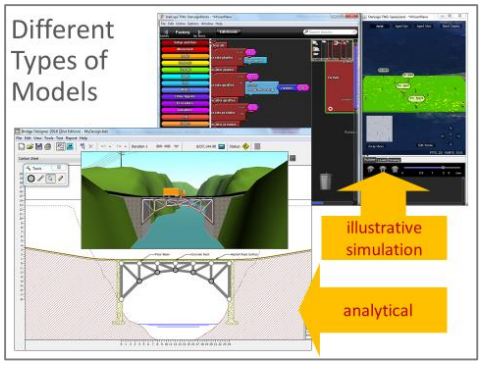
In this exercise participants are forced to rely on local communication. Agents in StarLogo are like blindfolded, whispering mice. They react to agents in their immediate environment only. The most common trigger for actions is a collision. We might call this 'bumpy programming'! Different patterns emerge from systems employing only local communication as we shall see through in further exercises.

Categorising Models

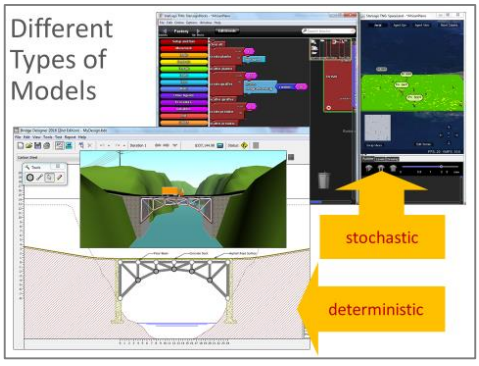
We started off by contrasting ***centralised*** models with complex rules and ***decentralised*** models, such as our simple eco-system. Another way of contrasting different models might be to consider their purpose. A model bridge can be used to ***analyse*** the precise effects of using thinner material for example. On the other hand, the sort of models we build in StarLogo are ***illustrative***, designed to illuminate the fundamental properties of a particular system. They do this through simulations, which can be run over and over again to observe the behaviour.



The behaviour of our simulations can change each time they are run. This is because they rely on local communication and have randomness built in. They are ***stochastic*** models. By contrast the bridge model is ***deterministic***. Change a parameter and the effect will be the same each time.

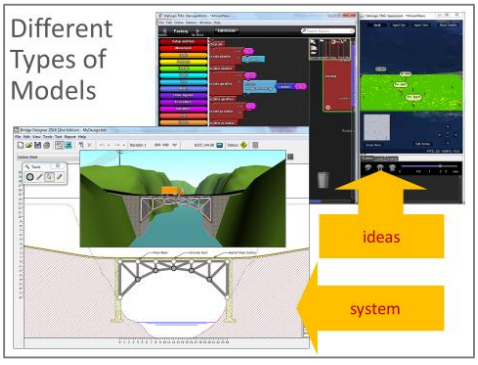


One final way of contrasting different models uses categories developed by Stanford biologist Joan Roughgarden. She talks about ***systems models*** which try to capture ever greater detail for a particular process. The more detailed the model, the more accurate the representation and the greater the complexity. She contrasts these with minimal ***ideas models***, which seek to capture just key essences to represent an idea.



Our simple eco-system is a case in point. It operates at a high level of abstraction. It captures the idea of food growth and consumption of both animals and plants but doesn't attempt to specify how things are eaten or reproduce. It operates at the level of ideas.

We have taken some time to illustrate these key properties of agent based models. As we move through our framework from using to exploring and finally building models, these ideas become more important. Later, when designing their own models, students need to keep them to the fore.



The key to ideas models is to keep them simple. That makes them accessible to young pupils and often provide ways to illuminate ideas they learn about in other scientific fields, particularly those relating to

natural science. These can provide motivational contexts, particularly for girls.