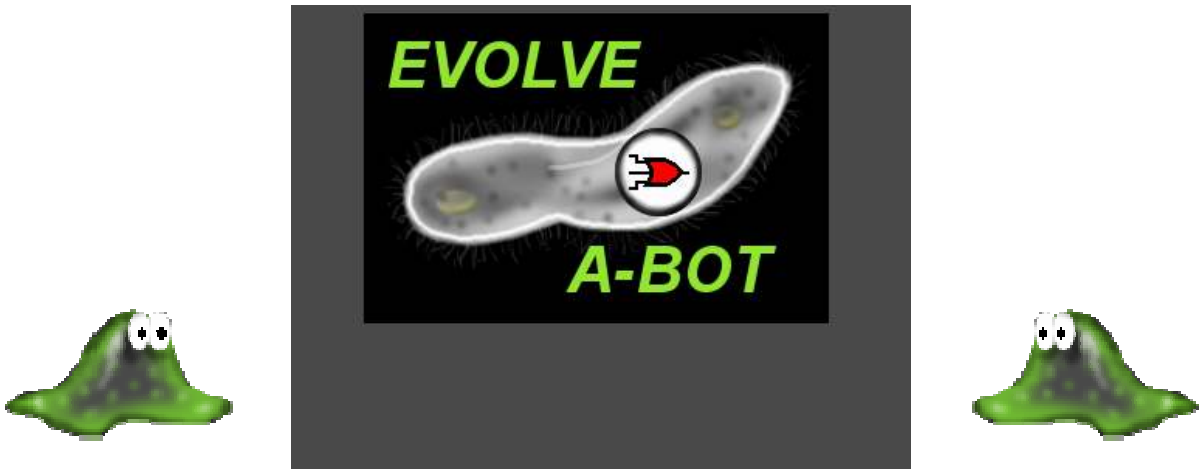


Game Manual

Have you ever wanted to program robots? Are you curious about how simple organisms like germs and insects can know what to do? How did such complex operation come to be? This game will show you how random numbers can produce serious and complicated programs that actually perform tasks, without anyone controlling anything.

The setting is a mystery building and starts in a locked room. You must figure out how to open the door. On the way through the building you discover things and solve some mysteries, including the operation of the Evolver, a machine that creates simple artificial life forms and uses evolution to create complex programs.

You will find out how to use the programs create through evolution to guide and operate robots and get them to do things for you. Hidden everywhere are clues and things that you will need. In the later stages you will encounter problems that require some thinking and the use of robots.

Let's cover the operation of the game first. Here is a list of keys and how they operate. Later we will look at the basic premise and game play.

Basic Principles

1. MOVING AROUND

Most programs follow the same general key functions. Moving around in the game is controlled by the arrow keys. The up arrow is forward, the down arrow is backwards. The left arrow turns you left and the right arrow turns you right. You can hold down more than one key at a time and move around that way.

2. DOING AND GETTING

The space bar is the key that makes actions occur. If you are near an object, hitting the space bar will pick it up. You might also hit the space bar to put it down. You can operate controls such as levers and buttons in the same manner.

3. BUTTONS AND CONTROLS

The space bar will cause some buttons to operate but when you have many buttons on screen, there has to be a better way. You can use the mouse and click any button. Some will always work, and some will only work in certain cases. You will be able to operate most controls if you can see them.

4. HELP MODE

In general game play you can get help by pressing F1. The arrow keys will guide you around once inside the help menu. ESC will take you out of the help menu. In Evolver mode help, you must click the HELP button again to exit.

5. CONTROLLING ROBOTS

As the game moves along, you will get control of various robots. Driving a robot around is simple enough. The WASD keys operate the robot in the same way that the arrow keys move you around in the game. "W" moves the robot forward, "S" moves it backwards. "A" turns a robot left and "D" turns a robot right. However, you must have control of a robot to do this.

When you charge or program a T-Unit (that odd shaped module that every robot has), you get control of the robot when you install the T-Unit. You can see that each robot has a slot on it that is red. That is where you install the T-Unit. You can see a T-Unit in the image to the right. You can grab or insert a T-Unit by using the space bar when you are close to a robot.

A robot that you take control of will turn light green. When you release the robot, it will change back to its original color. How do you take control of a specific robot? Type its number. Each robot has a sticker on the side with a number and a name. If you type in "3", and you have control of robot 3, it will turn light green and do whatever you direct it to do.

So, get the T-Unit, charge it, install it in the robot, and type the number of the robot. If it can be controlled, you will now have control of that robot. Later you can put your own programs in the T-Units and make the robots do many other things.

6. EVOLVER MODE

The Evolver has many controls but is actually simple. The Evolver is no more than an arena for tiny artificial organisms to move around and do things. Use your mouse to control the Evolver functions. You will find HELP and INFO buttons that will explain a lot.

In the Evolver arena, you can have ten artificial organisms called FLOMs. FLOM stands for Finite-Loop Organic Machine. Each is a little robot that has a program made of random numbers. This doesn't sound very useful but some of the random programs will actually do something. It is up to you to decide which looks most promising and to select them for further development.

In the real world the choice would be made by survival. If you have a million tiny identical germs, and the environment kills off most of them, only those that could make it under those conditions will reproduce. That is what shapes the organisms. Those with "programs" in their genes that don't work properly are "erased" by nature. Those that can get food, avoid danger, and reproduce will make copies that have the same genetic program.

The Evolver has a MUTATE function that will take the organism you select and make ten copies of it, but with a difference. Each copy has a single mutation in it. This means that little changes will be made with each generation, just as nature does. In the next generation the most-fit FLOM will perform a little better, live longer, or reach a goal. That is the FLOM that you will want to choose.

In the end, you will load the FLOM program of your choice into a T-Unit and put it in a robot. The robot will then be able to run your program and see if it works!

Starting a New Game

At the start of the game you will see a screen like this. You can start a new game or load an existing game. You may also ERASE an existing game and then your new game will start in that save slot.

Any progress that you save can be placed there when you finish your game. Just click a button to load or start a new game. You can pick up just where you left off this way.

When you finish playing, you can hit ESC to get to the SAVE screen. In this mode you can save your progress at any time. You can continue playing or quit from there.

Challenge Levels

FLOMs are simple creatures and need slow and careful progress to become complex and useful. That sort of change does not happen all at once. It is very, very unlikely that a full-blown complex program will emerge from nowhere. Instead, small pieces of the whole will emerge far more easily.

Your FLOMs must learn to move first. After that they will learn to seek light so that they, like plants, can charge up and have some energy. FLOMs can starve, just as a robot's batteries can run down. If you have some FLOMs being tested and they die, you will hear them gasp and expire. You can always give them a little reprieve and recharge their life by hitting the RESET button on the Evolver screen.

FLOMs have to learn movement, light seeking, food seeking, and other skills one step at a time. Each level opens more abilities for you and increases the skill of your FLOMs. After a few steps, you will have many software commands to give to your robots. This will be needed to get through the warehouse, a place where your robots will be tested.

Once you find a FLOM that looks promising, you can choose to MUTATE it. Go to the Mutator and use the arrow keys to select a FLOM that you want to reproduce. The Mutator will make ten copies of your FLOM with a mutation in each. These new FLOMs will then go to the arena where they will compete.

When you have all your basic skills in place, you will be capable of sending your robots off to handle jobs that are too dangerous for a mere human being!

Which FLOMs do I choose?

That's pretty simple. Choose FLOMs that live longer or move better. Choose the ones that reach the goal or seem to be drawn to what they are seeking. Success is the only rule here. Remember, if a FLOM can't get around well, you are not going to be able to tell if it is capable.

Finite State Machines

Years ago, a man by the name of Alan Turing came up with the idea of a state machine. This is a logical device that has a small number of well-defined states that it can be in. Each state is rather like a stepping stone and following a very simple set of rules, it performs basic tasks and then leads you to the next stepping stone.

FLOMs are just state machines that are filled with random numbers. Each state has four parts to it, and does not require complex instructions. Imagine a game that consists of stepping

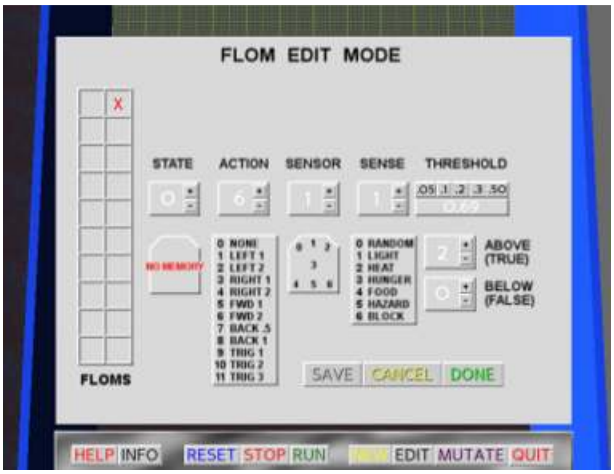
from one square to another, with each square containing an instruction and a decision. That is exactly how a state machine works.

The instruction might be “turn right” or “move backward”. Any simple instruction could be used. When you arrive on a square, you immediately perform the instruction that is there. Then the square has a sensor and a measurement. You would be expected to use the sensor and take a measurement of something. Typical measurements could be temperature, light level, or even a random number (like tossing the dice). You would do this right after executing the instruction.

Once you have taken your measurement you must decide what to do. In this case, you compare the measurement to a threshold value. If the threshold is 45%, then a reading that is 45% or more is “TRUE” while a reading that is less than 45% would be “FALSE”.

If you get a “TRUE” outcome, you will go to the next square that it tells you to. If you get a “FALSE”, you are told another square to go to. When you get there, the process starts all over again. You execute an instruction, take a reading, compare it to a threshold, and go to another square.

With as few as four or five such states or instructions, you can get a program that does something pretty sophisticated. As long as you have a small set of instructions and some sensors to measure your results, you can make a program that does something useful.



Using the Evolver EDIT mode, you can see the Turing states in your FLOMs. Select a FLOM with the arrow keys and then you can step through the states.

Each state has an action, a sensor, a sense, a threshold, and the TRUE/FALSE results. The actions are numbers and each stands for a specific action that the FLOM can take. The same is true for sensor and sense data.

Testing your FLOM Programs

How do you test four FLOM programs? You load them into a cartridge from the Evolver. When you have inserted a T-Unit in the Evolver, you can store a FLOM program in it. In EDIT mode you will see a red cartridge outline on the screen. Select the FLOM you want to test using

the arrow keys. When you have it, click the SAVE button. It will be lettered in red when the T-Unit is in the Evolver. Otherwise you will not be able to save your program.

Exit the edit mode by clicking DONE. Now click QUIT in the arena to get out of the Evolver. Remove the cartridge and put it in a robot. The robot now has your FLOM program in it. When you hit E (for EXECUTE), the robot will run off to test your program in an arena just like the Evolver. This is TRIAL MODE.

At the end of TRIAL MODE your robot's score will be shown. If it has performed as it should, you will complete the level and be ready for the next FLOM challenge. The robot will return and you can use the cartridge again to try the next program.

Once you have passed all the trial levels, your FLOMs will have figured out all the programs to control movement and sensing and seeking. This gives you all the tools you need to program and control your robots!

At the end of the challenges, you can then create robot programs at will and use them in a larger arena. But you will have things to do before you can access the large arena. That is where your robots will come in handy.

FLOM Senses and How They Work

Each FLOM has seven "sensors" similar to eyes. Each sensor can measure different things. The measurements range from zero to one (0 to 1). A reading of 50% is 0.50, while a reading of 85% is 0.85. If nothing is sensed, the sensor will return a zero.

There are seven unique senses that the FLOM has. Each can be detected from any sensor. You can choose a specific sensor and type of sense, or you can just use whatever program the FLOM has when it is made.

The senses are "random" which is like flipping a coin or throwing the dice; "light" which detects how well-lit the area is; "heat" which detects the temperature, "hunger" which is just how hungry your FLOM is; "food" which is like the smell of something to eat; "hazard" which is a general threatening condition, and "block" which senses obstacles.

Sensor 3 is located in the middle of the FLOM and is a general sensor while the others are at the edges or corners of the body and are directional. Sensor 0 is like a left front "eye" which will only see what is in front of it. Sensor 2 is like a right eye.

When your FLOM is running around, it can sense the overall light or heat level by reading sensor 3. It can look for a light source in a given direction by choosing that particular eye. This allows it to find where something is and either head for it or avoid it. This is the same

as hearing a noise to your right and because your ears detect that it is louder in one direction, you can tell where it is coming from.

Creating your own FLOMs from Scratch

This is an important step. If you look at the FLOM programs you will soon figure out exactly how they work. That means that you could edit a FLOM to give it a nudge in the right direction. Or, you could simply make a FLOM program up by planning each instruction for it.

This is true programming. You can literally take a naturally-evolved program and put your own finishing touches into it. You might enjoy thinking through each step of the process and truly understanding what it all means. Later, as you acquire the instructions, you can put together a standard program for your robots that combines the skills that your FLOMs have demonstrated.

Playing the Game

First, activate the robots and the evolver. Find secrets by using the floor mats. Sometimes you are unable to go somewhere but a robot could. Try it! Force fields will stop you from passing but they don't affect robots at all. Learn to operate your robots and use them as tools. Mats will trigger things (and you will hear a confirmation sound when this happens!)

When you have the Evolver running, use your FLOMs to evolve some programs. To try them out, load them into a T-Unit and plug it into a robot. The robot will accept the program and run off to an arena to try it out. This is Trial Mode and where your programs are judged to be fit- or not. The "E" key executes the program in your robot, "Q" will abort the test.

After you have achieved your goals, things will happen. Each goal gives your FLOMs and your robots more capability. In the end, you will have motion control, seeking and avoiding, and good, general instructions that can make your robots perform for you.

Winning the Game

You can expect to find hidden doors and sometimes force field barriers that you cannot get through. There are other machines in the building. You will find them as you play. Some are not so friendly. You are going to need your robots to help you defeat them and solve problems. Once you have your robots doing what you want, and you have confronted the final machine, you will have taken control of the building and won the game.

Your goals are simple. First, learn how to make smarter FLOMs. Second, put the FLOM programs to the test in your robots. Third, get your robots to do as you want. Fourth, take back control of the building by defeating the machines. Your programs can do it for you.

Remember, the things you learn here can be used in the real world. In fact, you can save your FLOM programs and they could be put in actual robots with very little effort.

What's next?

In the real world, most industrial robots are programmed using specialty languages or "ladder logic". What you can learn here is some basic programming concepts. You will also see how random numbers can actually create useful, orderly structures.

If you save your FLOM programs, you could in theory put them in a robot and watch them actually work in the real world. This program is the first step. We are creating the software "hooks" to make that happen. Using 3D printing and low-cost processor cards, you will soon be able to literally print a FLOM and install a processor, then use your programs from this game!

Have fun and enjoy this game. We hope that you learn something interesting in the process.

A Primer about Evolving Software

It may seem ridiculous that a very simple program made of random numbers could be smart enough to seek a heat source or avoid danger. These same instructions are deeply built into every living thing. Let's see how something as complex as a living microbe or an insect could be "programmed" by random chance.

Suppose we created a million FLOMs from scratch and we loaded their programs into real robots. These robots would then act in the real world and we could see which ones would perform well enough to be useful. It's pretty clear that if we had a six-state Turing machine and it used the exact format that we have here, we would have a good chance of getting something pretty close to a good solution to light seeking or heat seeking very rapidly. Why?

Because if we have six states per program, and each had 10 possible actions and seven sensors with six different things that could be detected, and we had ten threshold values, then the product of all this would yield almost a million possible unique programs. In other words, there is a pretty good chance that if a solution can be created in a six-state machine, we will probably have it or something very close to it in the batch of random programs. And, it's not unreasonable to assume that a "close enough" program would work pretty well in some cases.

So if we "threw the dice" a few times and made an entirely new random set of programs, we know that we will find a solution that is pretty close right away. Now, what if we chose a program and tested it? We stand a good chance of seeing the working program emerge very rapidly.

In other words, in a few tries we would probably have a very close solution in our hands in just a few minutes. All we would have to do is to drop the programs into some robot hardware for testing and see which robots did the best job. We are almost guaranteed that a solution will be handed to us in the mass of random numbers.

That is what Evolve-A-Bot is all about. And, if we make a few choices or edits at the outset, our odds get much better. In a matter of minutes we could have a solution for making a food-seeking robot. But because so much of this is determined by chance, there is no guarantee that you will find a solution. In theory you could throw a dice cube twenty times without getting a one, but in the real world that is unlikely.

In the same way, there can be many solutions that are close to ideal and a good number that are fair, and all we need to see is that the program we arrive at does something that looks like intent. A random program will sometimes do something that appears to be what we want, and if we mutate it a few times it could very well get closer and closer to the proper actions.

This works because we only allow programs that appear to be functioning at least partially to be refined. Any program that does nothing or is counter to what we want "dies" or is eliminated. So what we see is that our programs appear to be "getting smarter" and that is exactly what happens in nature. Organisms that can feed themselves have a chance of not starving. Organisms that can move around have a better chance of avoiding danger or finding food. Any organism that can't eat or move will die very quickly.

And, faster rabbits can evade the wolf. Any FLOM that is a little more capable will have a better chance of surviving because you will see how it performs and might select it. But keep in mind that more complex programs in later levels may take longer to evolve.

But wait! Nature does not choose!

Actually, it does. Nature does not look at things and go "hmm, that's better." But what does happen is that animals will starve or eat, they will escape or be trapped, and in the long run the hazards of the environment make the choice. There is no mind behind the process, just the laws of chemistry and physics. A falling rock or a drought can be just as final. By the same token, finding a pond of clean water or a bit of vegetation for shade can make all the difference.

So, while we are personally choosing the FLOMs that live or die, it is just as easy to make the program do the choosing. A piece of software can act like a world and present hazards and opportunities. We could just as easily simulate heat and cold, rain and disease, and use them as the tools to choose what lives and thrives.

The bottom line is that this artificial process of evolution is just as legitimate as the one that is all around us. The only difference is that we are using "selective breeding" to make the FLOMs that we want, just as farmers over the thousands of years have produced the crops that we eat today by choosing the best, largest, and tastiest.

Another Look at "Unnatural Selection"

In the real world there are constant and often unpredictable dangers. Poor weather, predators, lack of food, and drought are just a few we might mention. Nature uses a simple means of selection- survival. Animals and plants that survive are capable of making offspring, and that is how the ability to adapt and survive is passed along to the next generation.

In Evolve-A-Bot you are the selector. You get to choose what lives and passes its abilities along. Once you understand a couple of simple rules, you will be able to shape and guide the process of evolution in any way you wish. Here is an example.

In the very first challenge level you must find FLOMs that can move around effectively. That is easy to do. Those that stay where they are, run in circles, or crash into the wall are probably not good choices because the rules that tell them to do that will remain in their offspring. What you want is something that gets around well.

Look for FLOMs that seem to move more than the others. Pick the best one and use the Mutator on it. Now the arena will be cleared and all the other FLOMs will be gone. Only the ten new FLOMs that emerge from the Mutator will remain. These FLOMs will then be tested and you can see which ones move the best. Your number one goal is to get highly mobile FLOMs and two things will help you know which to choose. Number one, which ones live longest? Number two, which ones move the best?

Those choices will determine how well the new FLOMs will perform. In as little as three generations you may have a batch of well-mannered, highly movable FLOMs and that is your first goal. But since this process is random, you may have to spend more generations getting what you want. This will become apparent when you get to more complex levels.

Complex programs take longer to evolve. And, nothing prevents you from going into edit mode and making certain that your FLOMs have the right instructions to help them move, or eat, or do whatever you want them to be doing. In fact, this is an important step in truly

understanding the process. Your edits will be amplified over time as they carry to all the new offspring of your FLOMs.

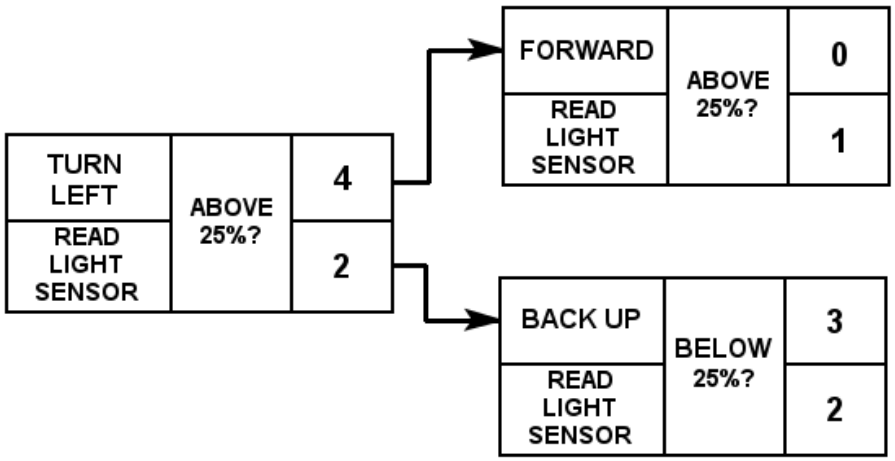
Editing FLOMs and Other State Machines

The big issue is how to do the editing. The Evolver has an editor that allows you to see exactly what instructions your FLOMs are following. If you know the instructions, you can make your own programs up or edit the programs that arrive from random numbers. Let's take a close look at EDIT mode and see how it works.

First, the FLOM program is a set of states, as mentioned above. Each state has an action, a sense, a choice that is made, and a pair of states to choose from. You can become familiar with the parts easily.

Think of states as being like stepping stones or the squares of a board game. By following the rules, we can choose the next square to move to. Now, let's make this much simpler than a board game. Imagine that there are only six squares in the game. Now you have a very limited map to work with and that makes life easy.

The first square might tell the FLOM to turn left. This might be followed by an instruction to check how bright it is (light sensor). If it is dark, you might want to move backwards but if it is light you might want to move forwards. That means that you must choose which instruction is next, and the two instructions that follow will two new squares you could move to.



An example of states in a FLOM program

One square will have “move forward” as its instruction followed by another look at the light level. The other will be “back up” followed by checking the light level or maybe picking a random number. In either case, the next choice is being influenced by light level. See the illustration above.

It can be helpful to make a diagram of what you want to try. Using the editor, you can create that program by putting your own instructions in the states and choosing your own sensors. In the end your program will be whatever you would like but with an extra step. You can give it to the Mutator and see what emerges.

This is the power of evolving your software. Any idea you have can be entered and it will mutate and spawn offspring programs that are almost exactly what you wrote, but the programs will change slowly and become a mix of your intent and the random nature of the program.

In other words, even your ideas can be mutated and may lead to something entirely new. Think about that.

So if you have an idea for a better heat seeker, this is where you can put your ideas and see how they work. And, you can allow them to mutate and become some new version of your process which may be better or probably worse. But some surprises can emerge here.

It’s Just Not Happening

You have a FLOM that goes for it immediately and produces excellent results. In the arena with the robot however, it floats about in a daze, never even looking at the proper course. What happened?

It is quite possible that your FLOM literally wandered into the proper solution and just looked like it knew what it was doing. It’s called dumb luck. It can fool you into thinking you have a winner when all you have is random chance.

If you have problems getting your FLOMs to converge on a solution, you can always throw them away and start with a new group. Just click the NEW button and ten new FLOMs will appear. But remember that the other FLOMs will vanish, just as if they never existed at all.

Again, a fresh group of FLOMs should be selected first for long life and movement ability. Later you can mutate them and see how well they work. Be patient; life on Earth took about four billion years to reach this point. FLOMs evolve many millions of times faster than living things but they are simple and have an entirely different environment than the real thing.

Don't be afraid to experiment. There are no rules here except success. After you have grown and refined your FLOMs, load them into a robot and watch the action in the arena. You will surely find something there that will be interesting and helpful.

Finally, you will face challenges in the different rooms of the game. You must create robot programs that can defeat each challenger. Your robots must seek solutions and evade dangers. That will demonstrate that your evolved software pieces can be put together and used in real situations.

"Organic" solutions are not the same as directed solutions that human beings tend to create, but they can be effective just the same. Instead of heading right for the heart of a problem, your FLOM programs might veer off to the side for a while. In the end however you will see that they can handle things in a way that traditional machines do not.

Making up FLOMs from scratch

Here is where we get to the heart of creating state machines. This will enable you so that you may create a FLOM program with intent.

All of the instructions that FLOMs have are "deterministic" or without choice, with the exception of one. That is the random number instruction. For example, if your FLOM executes a "move forward" instruction, that is exactly what you will get. However, the random number instruction creates a random value that ranges from 0 to 1. There is no way to predict what the value will be. This means that you can put movements or decisions in your FLOMs that are the equivalent of flipping a coin or just making a guess.

So what if I want a FLOM that moves ahead some random number of times? I can create a state that has "move forward" as the instruction, then pick a random number, and finally decide if we need to do it again. For example, if you pick a threshold of 0.5 and then generate a random number, there is a 50-50 chance that the random number will be greater than 0.5. If you set the threshold to 0.1, there is a one-in-ten chance that the number will be below 0.1 and a 90% chance it will be greater.

What this leads to is that with a threshold of 0.1, about one time in ten you will get a number smaller than that. If you have a state that takes a step forward each time it executes and then generates a random number with a threshold of 0.1, you can loop it back to itself so that it will take an average often steps before doing something else. It may take one step or thirty, but on average you can get a nice ten-step walk.

This means you can create a program that takes a sort of random walk instead of a pre-determined, unchanging walk. Of course it will be affected by whatever the FLOM is sensing. So let's look at an example.

If you want a FLOM that goes forward until it senses light, then turns to it, you can do it in a pretty straightforward manner. First, you want a state that determines if light is present. If it is above a level (such as 0.3) then you want to turn to that light. Otherwise, you want to cruise around looking for a light level that interests your FLOM.

So one state (let's call it zero) will move your FLOM forward until it detects a light level greater than 0.3. Then it will go to a state that makes it turn to the light. Now you can go back to your first state and cruise some more. If you wanted to refine it, you surely could. But two states can create the cruise and the detect command, at which point you could then choose another course of action that homes in on the light source, and finally locks on and sits there.

State zero: Forward, read light sensor, if > 0.3, go to state one, otherwise state zero

State one: Turn right, read light sensor, if > 0.5, go to state two, else go to zero

As you can see, this two-step state machine causes your FLOM to locate a light source and if not, keep searching until you find one. Otherwise, state two is executed which might do literally anything.

The most common solution is to generate a random FLOM, mutate it a few times, and then refine by editing the states. However, you can definitely make your own and then evolve them just as any other FLOM could be treated.

Have fun!

A simple light-seeker example

State 0: 6 – 0 – 0 – 0.1 – 0 – 1
State 1: 3 – 1 – 1 – 0.2 – 2 – 0
State 2: 2 – 3 – 1 – 0.5 – 2 – 0