Sodarace:
Exploring Evolution with Computational Thinking

Age group: 8 – adult
Abilities assumed: None
Time: 30 minutes
Size of group: 1 upwards

Focus
Computational Thinking, Algorithmic Thinking, Computational Modelling, Abstraction, Evaluation, Generalisation.

Syllabus Links
This activity can be used (for example)
• as a general introduction to applications of algorithmic thinking in the form of computational modelling from KS2 up.
• as a general introduction to evolution from KS2 up.
• to show how computational models can be used to understand real-world phenomena from KS2 up.

Summary
You demonstrate the Sodarace Kiosk – a modelling world where you can engineer racing creatures called amoebas, but also let your designs evolve over many generations in a series of races where only the fastest survive to breed and so race again. You explain the process of evolution. The class then explore how it works by evolving their own races over different tracks. Will anyone evolve an amoeba that flies off the cliff and across the line without touching the ground?

Technical Terms
Algorithms, computational modelling, algorithmic thinking, computational thinking, evolution, natural selection, frequency, amplitude, genetic algorithms, local optima.

Materials
Sodarace Kiosk Demo video
Sodarace Kiosk installed on all classes computers (www.sodarace.net)
What to do

In preparation
Download the demonstration video and the Sodarace kiosk software, installing the latter on computers for the class to use (The software can also be used in pairs at a single computer).

A demonstration
Either show the demonstration video of the Sodarace kiosk being used or demonstrate the Kiosk directly by working through an example similar to the video.

It is a race between different creatures. Can you create an amoeba creature that can win the race? How fast can you make the creature go and will it win the race? Can evolution then create something even faster than you could engineer just with random changes?

First you select one of the three pre-designed creatures.

You then pick a race-track for it to race over (flat, slightly hilly and over the cliff).
Next run a race. See how well your chosen creature does. Which creature gets to the finish line first?
Now you can try to engineer a better creature. By moving the sliders you can give it more or less points and lines either making it sturdier or slimming it down. You can change the **frequency** of energy boosts it gets (i.e., feed it more or less often) and change the **amplitude** of energy boosts (i.e., give it bigger or smaller meals each time). In doing so you create a new creature that could be faster or slower – enter it in the race to find out.

Alternatively you can try the automatic process for creating a new creature. It uses natural selection. The initial creature has children (randomly mutated versions of itself) – you can see the children appearing at the top of the screen). All the children are raced against it, and only the fastest survives. If any child is faster then it becomes the parent for the next generation. This is repeated for as many generations as you like.

The bar chart has one bar per race. It shows the time taken for the fastest to win the race that generation, so when the bar gets smaller a faster child has been found. That child will become the new parent. When you have evolved a child that is fast enough for you, then run the actual race again and see how it does.
Over time the random changes build up to give faster and faster creatures (some times at least). That is how evolution works. The slowest are eaten by predators, or are too slow to catch anything to eat so die without children. The fastest survive and the random changes build up over generations to make them faster and faster. Actual evolution doesn’t just work on how fast an animal is though – it works on anything that is the difference between survival or not.

**Running Races**

The class can now be given time to use the interactive version. Can they create a creature that wins the race? Can anyone create a creature that ‘flies’ off the cliff, crossing the finish line without touching the ground? Talk to them about how well evolution works as they run the program.

As the changes are random, it sometimes might take a long time to find a faster child, whereas at other times one is found straight away. If evolution seems to be making no progress after lots of generations, have the students stop it and tweak the creature themselves then try again to see if it makes a difference. Sometimes evolution hits what is called a **local optima**. A creature is better than anything similar to it, so small changes aren’t enough to find an improvement even though with a slightly bigger change a faster creature might result.

**Rounding Up**

Finally, pull every one back together and ask them to describe what happened. Summarise again how evolution works. Explain how the class have been using a computational model – an algorithm simulating evolution. It is a simplified version (using abstraction) e.g. here we only have a 2D world, the survival test is just a race rather than really surviving, and so on. It shows how by using **algorithmic thinking** in the form of creating computational models we can explore how a scientific idea
works and whether the mechanism suggested by scientists has the effect they claim. We have therefore been doing a form of **evaluation** but of the scientific theory using an algorithmic version of it.

Computer scientist’s have also **generalised** these ideas about modelling evolution, to create a new way to solve problems. Called **genetic algorithms** they are a way of finding coded solutions to problems using processes based on evolution. You create digital DNA to represent possible solutions. It is just a set of values that represent a possible solution to the problem, like the 4 values of our racers: number of points, number of lines, frequency and amplitude. A quick and easy fitness test is needed (like the race) to decide which are the best solutions of any generation. Then small random changes are made to a set of solutions, the best survive to the next generation as decided by the test, and the process is repeated over thousands of generations. Gradually from all the small random changes better and better solutions emerge.

Genetic algorithms have been used for problems like coming up with good classroom timetables, designing cars, finding routes, good strategies for investing money and also as a way of creating programs that can compose music or make art.

**Variations and Extensions**

**Sodarace your own creatures**
Create different creatures and race them against each other using the full Sodarace rather than the simple kiosk version as here (www.sodaplay.com)

**Other Sodarace and Soda constructor activities**
There are a variety of other activities for schools and cs4fn articles on Sodarace. See http://www.cs4fn.org/alife/sodaindex.html

**Read more about computer science and evolution**
There are a series of articles on computer science and evolution, and genetic algorithms at http://www.cs4fn.org/evolution/
Links to other activities

Get the following activities with links to biology from teachinglondoncomputing.org

**Soda Constructor: Physics**
*Create creatures in a line and dot drawing package, then switch on the laws of physics and see them come to life.* Explore gravity, friction and springs and how creatures can develop many different ways to move about out of a few laws of physics.

**Brain-in-a-bag**
*Create a brain out of rope, tubes and kids that plays Snap.* Demonstrate how neurones work, as well as how computational modeling can help us understand the world, and how biological processes can lead to new forms of algorithmic thinking.

Live demonstration of this activity

Teaching London Computing give live sessions for teachers demonstrating this and our other activities. See [http://teachinglondoncomputing.org](http://teachinglondoncomputing.org/) for details. Videos of some activities are also available or in preparation.

Acknowledgements

This activity sheet was funded by a grant (ESD23/2014) from the Department for Education, with support from Google. Teaching London Computing is funded by the Mayor of London. Sodarace was created by Soda Ltd in collaboration with Queen Mary University of London. It is based on the former’s BAFTA winning Soda Constructor.

Use of this material

[Attribution NonCommercial ShareAlike - "CC BY-NC-SA"](http://creativecommons.org/licenses/by-nc-sa/4.0/)

This license lets others remix, tweak, and build upon a work non-commercially, as long as they credit the original author and license their new creations under the identical terms. Others can download and redistribute this work just like the by-nc-nd license, but they can also translate, make remixes, and produce new stories based on the work. All new work based on the original will carry the same license, so any derivatives will also be non-commercial in nature.