University of Kent

Kent Academic Repository

Brown, Neil C.C. and Kölling, Michael (2012) *Position Paper: Programming Can Deepen Understanding Across Disciplines.* In: Addressing Educational Challenges: the role of ICT (AECRICT 2012), July 2nd - 5th, 2012, Manchester Metropolitan University, Manchester, UK.

Downloaded from

https://kar.kent.ac.uk/33881/ The University of Kent's Academic Repository KAR

The version of record is available from

This document version

Publisher pdf

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in *Title of Journal* , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

Position Paper: Programming Can Deepen Understanding Across Disciplines [DRAFT]

Neil Brown, *nccb@kent.ac.uk*
School of Computing, University of Kent, Canterbury, Kent, UK

Michael Kölling, *mik@kent.ac.uk*
School of Computing, University of Kent, Canterbury, Kent, UK

Extended Abstract
Computer programs can be used to support learning in various subjects: In particular, computer simulations can be used to explain science subjects such as Physics, Chemistry, and Biology (as well as wider subject areas such as Economics, Geography and Music). Educational simulation programs are often used as a black-box: Students are given the program to play with and are told about the principles underlying the simulation, but they are given no opportunity to relate the computer program to the knowledge being taught. The subject of Computing, in contrast to ICT, enables students to read, understand and modify program code and algorithms. These skills can be used, in conjunction with open-source simulations, to allow students to directly read and understand (and potentially modify) the key parts of the simulation programs that they are using. We believe that this use of programming skills can allow for a deeper and more direct understanding of the subjects under investigation, using Computing to support learning in the same way that Mathematics supports the learning of subjects such as Physics.

Keywords
Greenfoot, Simulation, Education, Programming

INTRODUCTION
Information and Communications Technology (ICT) in schools takes various forms: from smart whiteboards and graphing calculators through to software packages (desktop publishing, word processing, etc) and specific computer-aided learning packages (e.g. for language learning). Several subjects, especially the sciences, can make use of educational simulations to support, complement or even replace practical experiments (e.g. the PhET project (2011) resources). In all of these cases, ICT is used as a black-box technology, with functionality that supports learning, but with no transparency – no visibility of the inner workings.

A key difference between Computing and ICT is that the former includes examining and altering how programs work, rather than simply using them (CAS, 2011). In this paper, we examine the possibilities that are opened up for learning if students are able to read and adapt code and are able to use Computing as a basis for learning, rather than ICT: What if the students could not only use computerised simulations of scientific concepts, but could also examine and alter the way in which the simulation is implemented?

In this paper, we explore several subjects and describe how Computing could benefit the teaching of some example topics in those disciplines. The examples will be given using the authors' popular Greenfoot (2011) programming system (suited to ages 14 upwards), which uses the Java programming language to support the creation of interactive games and simulations. The choice of environment is not central to this paper, and alternative environments such as Scratch (2011) (suited to ages 7—14) would function equally well.

BIOLOGY

Biology – in particular, ecology and the study of populations – lends itself well to illustrative simulations (e.g. Kölling, 2011). A classic example of organism behaviour that has developed into its own sub-discipline of computer science is the study of swarms and behaviours of creatures such as ants (e.g. Kölling, 2009a), inspired by the famous boids example (Reynolds, 2011). This swarm behaviour is usually expressed as a series of simple rules, which can be directly translated into a computer program: align with your neighbours, keep reasonably close to them, but avoid collisions. Understanding the behaviour of the swarm exactly corresponds with understanding the algorithm expressing the creatures' behaviour.

An example of how code reading could deepen understanding is the investigation of how drops of pheromones are being used by ants in the ants simulation. If students were able to also modify or write code, one could ask questions such as "If a pollutant were introduced into the environment that causes the sense of smell to deteriorate by 50%, would ants still be able to form paths?" This could easily be investigated experimentally by making small changes to the code. Another obvious task might be altering the size of a habitat in a predator/prey simulation to investigate whether habitat size influences probability of survival of species. In each of these cases, fairly minor modifications allow deepening of interesting insights.

Another example where simulation can be useful is in explaining natural selection. One Greenfoot example (Kölling, 2010) invites the human to play the role of a predator in an environment in order to observe the principles of natural selection. The scenario starts with a random population of frogs. The code is straightforward, picking a random frog, which will "reproduce". When a frog reproduces, it forms copies of itself, which have a small chance of having a slightly altered colour from their parent (i.e. a small mutation). This code can be understood fairly easily, and when run (with some code to *randomly* kill off old frogs) the simulation produces a random series of coloured frogs.

The user of the scenario is then challenged to "kill" as many frogs as possible (by simply clicking on the frog to remove it) in a short time frame. Played against a green background, the player will automatically click on frogs whose colour stands out against this background, and over a period of time, this evolves the frogs to become green, even though the code shows no such bias. Playing against a red background will instead breed red frogs. This combination of interaction, and understanding the lack of in-built colour bias in the code, can allow a good understanding and appreciation of the natural selection process. Furthermore, the students can alter the code to answer questions: what if the colour is no longer based on the parent frog, but is entirely random – do the frogs still converge to the colour of the background (and in the same amount of time?).

PHYSICS

The idea of using computer-based simulations for Physics is far from new. In particular, the PhET (Physics Education Technology) project (2011) is a successful example of producing simulations for Physics (and other natural sciences) that can be used for teaching a variety of ages. The PhET examples are delivered as black-box simulations: ready to use, but without the possibility to easily examine (or alter) the source code. One example from the PhET project is the "wave on a string" simulation, which simulates wave propagation along a string, with configurable damping. This exact example has been replicated in the Greenfoot system (Kölling, 2009b); in Greenfoot, the source code can be viewed, showing the key behaviour of the system in a few lines:

```
middle = (leftNeighbour.getExactY() + rightNeighbour.getExactY()) / 2;
newForce = (middle - getExactY()) * 2;
movement = (newForce + movement) / (1.0 + damping);
setLocation(getExactX(), getExactY() + (movement / 4));
```

This code is easy to read for someone with basic mathematical knowledge, even without much programming knowledge. This ability to see the exact inner workings can help to solidify the students' understanding of the underlying concepts. By altering internal simulation parameters or behaviour, students could experiment with characteristics of different materials, such as difference in elasticity.

MATHEMATICS

Graphical frameworks such as Scratch and Greenfoot are heavily dependent on trigonometry to function. The underlying mechanism to rotate images uses a trigonometric transformation, moving sprites a fixed distance at a given angle is a straightforward application of trigonometric laws. Thus, one easy way to present a practical application of trigonometry to students (who sometimes complain of mathematics' lack of obvious application) is to expose this functionality with the simple equations written into the program:

```
x += dist * Math.cos(angle);
y += dist * Math.sin(angle);
```

Many games include functionality, for example, to find the nearest object, or to react as a function of distance to another object. Since object location is usually expressed in Cartesian coordinates, this typically involves the use of Pythagoras' theorem. These examples can be used in a code reading exercise to uncover the mathematical foundation of simulations/games, or they can be used constructively, by setting students the challenge to extend a given program in this way.

CONCLUSION

This paper has outlined how knowledge of Computing, particularly programming, can be used to support education in other disciplines and provide deeper insight into the topic under discussion, while simultaneously practicing computing skills as a general tool for scientific discovery. We propose that if students are taught to read code (even if their ability to write code is limited), they can gain great benefits from exploring the code behind the simulations that are used to support their learning. This ability to read code need be treated no differently than the ability to read a mathematical equation (which is, in some respects, a similar skill) or the ability to cross-reference material.

Once students have the ability to read code, simulations can become more instructive by allowing students to see (and alter) how they work, and thus learn the underlying principles of the simulations – rather than simply using them as a black-box tool for experimental simulation. Such simulations can be constructed in environments such as Greenfoot or Scratch, and then shared for many teachers to use. Making the core code readable and understandable should be a priority, but this is already good software development practice.

The cross-disciplinary applications of Computing described in this paper are not restricted to the more obvious natural science subjects that we have chosen to focus on. Subjects such as Economics and Sociology can use simulations of the population in order to explore the effects of tax policy or the spread of information. A score in Music can be viewed as a deterministic program, ready to be executed – but Computing allows the exploration of procedurally generated music: What if the

score were not deterministic, but had defined choice points? We believe that Computing can support many other disciplines, analogously to Mathematics. These cross-disciplinary applications of Computing should be recognised and implemented by teaching basic code-reading (and, ideally, code-writing) to all students, then making use of these skills across the curriculum.

REFERENCES

CAS (2011) *Computing: A Curriculum for Schools,* Computing at School Working Group, http://www.computingatschool.org.uk/

Greenfoot (2011) http://www.greenfoot.org/, accessed December 2011

Kölling, M. (2009a), Ants Greenfoot Scenario, http://www.greenfoot.org/scenarios/1016, accessed December 2011

Kölling, M. (2009b), Wave-Lab Greenfoot Scenario, http://www.greenfoot.org/scenarios/1007, accessed December 2011

Kölling, M. (2010), Natural Selection Greenfoot Scenario, http://www.greenfoot.org/scenarios/1385, accessed December 2011

Kölling, M. (2011), Foxes and Rabbits Greenfoot Scenario, http://www.greenfoot.org/scenarios/3902, accessed December 2011

Reynolds, C. (2011) Flocks, herds and schools: A distributed behavioral model. In SIGGRAPH '87. pp. 25—34.

Scratch (2011) http://scratch.mit.edu/, accessed December 2011

The PhET project (2011) http://phet.colorado.edu, accessed December 2011

Biography

Dr Neil Brown is a Research Associate in the Computing Education Research Group, in the School of Computing at the University of Kent. He works on the Greenfoot project as one of the lead developers, and is a working member of the Computing At School (CAS) group.

Professor Michael Kölling is a Professor of Computer Science in the Computing Education Research Group, in the School of Computing at the University of Kent. He is responsible for the creation of the popular BlueJ and Greenfoot learner's development environments, and is a Distinguished Member of the ACM and a working member of the CAS group.

Copyright
This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/3.0/