THE NATIONAL CENTRE FOR COMPUTING EDUCATION

What is it, and how can it help?

SCRATCH 3

- Introduction to the new Scratch
- micro:bits and tablets
- New extensions
- And much more

GET YOUR PROGRAM INTO SPACE

Take the European Astro Pi Challenge

SPECIAL NEEDS

National survey findings explored

WHAT'S THE BEST WAY TO TEACH PROGRAMMING?

PLUS MAKING A 3D ANIMATED FILM • WHY COMPUTING HISTORY MATTERS • COOLEST PROJECTS 2018 RESEARCH • CLEVER CAT MACHINE LEARNING • MICRO:BIT AND MR BIT • DIETING WITH GENETIC ALGORITHMS
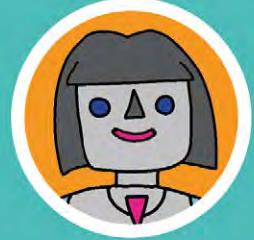
START A CODE CLUB IN YOUR SCHOOL!
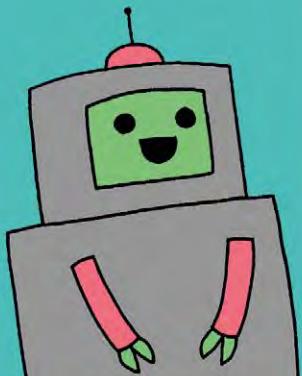
Code Club is a network of volunteers and educators who run free coding clubs for young people aged 9-13.

Our aim is to inspire the next generation to get excited about computer science and digital making.

"We use Code Club's fun educational resources to run a weekly after-school club for Year 7 and Year 8 pupils. The students benefit considerably from the extra challenge!"

Karen Dadd, Computing Teacher

☑ **Code Club is free**

☑ **Code Club provides step-by-step guides for Scratch, Python, HTML, and Sonic Pi**

☑ **Code Club helps children develop skills including logical thinking, creativity, and resilience**

We have over 6000 clubs across the UK teaching more than 80,000 young people to code—come and join us!

Find out more at www.codeclub.org.uk

HELLO, WORLD!

I t's so exciting to see how the new National Centre for Computing Education is taking shape as a collaboration between STEM Learning, Raspberry Pi, and the British Computer Society. This is a once-in-a-generation opportunity to ensure every child in England has a world-class computing education. Although the centre has only been established a couple of months now, there's already a great set of online and face-to face-professional development in place: Sue Sentance gives a great introduction to the project here, and keep an eye on **www.teachcomputing.org** for the latest news. There'll be lots more about the NCCE in the next edition of *Hello World*.

Computing At School plays a crucial role in the work of the National Centre, as a grassroots community of practice with national reach, in which teachers share their expertise, support, and encourage each other to become better and more confident.

The National Centre will be establishing 40 school-based hubs this year, but CAS's local network, now "CAS Communities", remains more important than ever.

We're leading with the release of Scratch 3 for this issue. The new version makes Scratch more accessible than ever, with great built-in tutorials running very happily inside the browser on tablets. It also extends the uses of native Scratch to applications like machine translation and robotics. I'm excited to see how these new tools get used within and beyond the classroom.

Miles Berry
Contributing Editor

FEATURED THIS ISSUE

SUE SENTANCE
CHIEF LEARNING OFFICER, RASPBERRY PI FOUNDATION
Sue is Chief Learning Officer, Raspberry Pi Foundation. An ex-teacher, teacher trainer, and academic, she is passionate about research in computing education.

MITCH RESNICK
PROFESSOR OF LEARNING RESEARCH, MIT MEDIA LAB
Mitch develops new technologies and activities to engage people (particularly children) in creative learning experiences. His goal: help everyone learn to think creatively, reason systematically, and work collaboratively.

CATHERINE ELLIOTT
ELEARNING CONSULTANT
Catherine works with the Sheffield City Council eLearning Service. She has spent the last five years looking at how to adapt the computing curriculum for learners with SEND. She is joint leader of the Sheffield & South Yorkshire Secondary CAS hub.

(HW) CONTENTS

NEWS AND FEATURES

22

SCRATCH 3 ON TABLETS
One school goes hands on

ENGAGING ALL STUDENTS
SEND survey results digested…

30

NATIONAL CENTRE FOR COMPUTING EDUCATION
All you need to know about a major new resource

68

CONVERSATION

REVIEWS

LEARNING

TUTORIALS & LESSON PLANS

The Raspberry Pi Foundation CEO Philip Colligan taking lessons from his son, Dylan

COOLEST PROJECTS 2019
DATES ANNOUNCED

A showcase for young innovators, Coolest Projects is simply inspirational. Rosie Hattersley reports

B ack in 2012, when Raspberry Pi was merely a great new idea garnering lots of positive praise, another excellent innovation aimed at inspiring young creators was also in its infancy.

Coolest Projects (**coolestprojects. org**) began in Ireland as a science fair and forum for like-minded inventors to meet and show off their ideas. Founded by CoderDojo volunteers Noel King and Ben Chapman, the event has gone from strength to strength, with thousands of young scientists and technologists attending

Dublin's annual Coolest Projects showcase each year.

This year, Coolest Projects came under the wing of The Raspberry Pi Foundation, with events in Ireland, the UK, and North America.

These locations will be hosting Coolest Projects fairs again in 2019, according to The Raspberry Pi Foundation CEO Philip Colligan.

Coolest Projects North America will be held at The Discovery Cube, Orange County on Saturday 23 March, while RDS in Dublin takes a turn to host on Saturday 5 May. Ahead of both these

showcases will be the UK event, now in Manchester at The Sharp Project on 2 March.
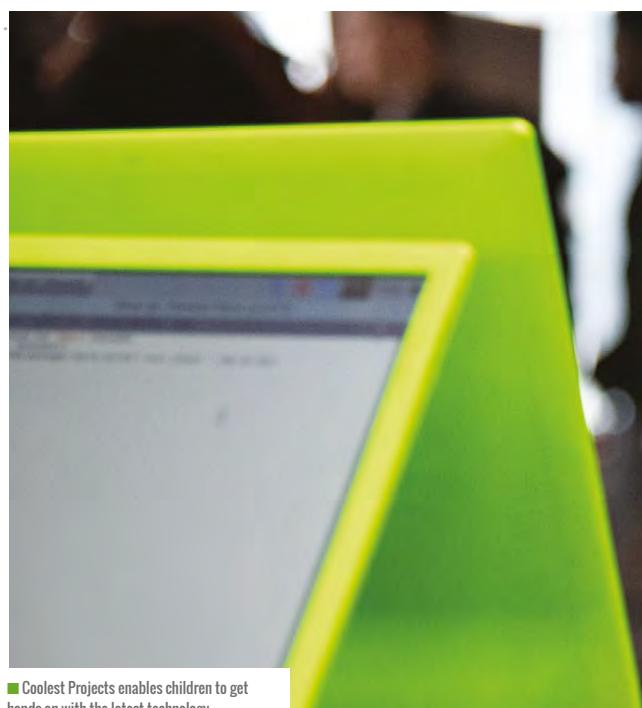
Philip says, "Coolest Projects is where thousands of young people showcase amazing projects that they've built using digital technologies. If you want to meet the innovators of the future, this is the place to be."

For him, the event isn't just about the creativity, innovation, and sheer effort that everyone puts in, but the sense of community. If that doesn't sound like the sort of event a Raspberry Pi fan might be involved with, we're not sure what does.

Registration for all three events opens in January. Milan, Belgium, and Bulgaria will also be holding community events that will run roughly concurrently with the Coolest Projects science fairs next spring. (HW)

■ During Coolest Projects, kids get to show off the amazing projects that they've built

■ Coolest Projects enables children to get hands-on with the latest technology

BLETCHLEY PARK
HOSTS GIRLS' STEM DAY

National Museum of Computing holds STEM event for girls. Bravo, says Rosie Hattersley

Bletchley Park played host to a hugely successful Girls' STEM Day in late November, creating a real buzz among participants and demands for more events of the kind.

Among the sessions were Raspberry Pi workshops held in The National Museum of Computing's (**tnmoc.org**) newly refurbished classroom. *The MagPi* writer Mark Vanstone witnessed students getting stuck into Python coding and Raspberry Pi-based Minecraft.

The female-focused STEM showcase was held at The National Museum of Computing, sited at the famous code-breakers' venue with the express intention of highlighting the breadth of career possibilities to students who may not otherwise have been aware of their aptitude for such industries. It's no secret that there's a significant imbalance in the numbers of males and females in STEM (science, technology, engineering, and maths)-based careers.

Untapped potential

Cybersecurity company Sophos was on the scout for potential computer scientists of the future. VP Ali Kennedy noted that more than half the UK population is female and that "much more female involvement will be vital as we move into the fourth industrial revolution". Meanwhile, Buckinghamshire High Sheriff Professor Ruth Farwell gave an inspirational talk in which she credited her own start on the road to a high-profile career to studying a STEM subject at university.

While many of the 80-plus participants on the day already had an ongoing interest in science and technology, lots of attendees were delighted to unlock their previously undiscovered talents, and

■ Students were challenged to complete tasks using Raspberry Pi computers

■ The Girls' STEM event was held at The National Museum of Computing, sited at the famous code-breakers' building

■ The RAF held a STEM Engineering Challenge with students working in teams

impressed observers with their teamwork and ingenuity.

The RAF held a STEM Engineering Challenge, in which students had to master unfamiliar equipment and work in teams in order to complete tasks. Challenge organiser Chris Mossman came away seriously impressed with the students' aptitude for logical thinking and problem-solving.

PJ Evans, a TNMOC volunteer and *The MagPi* contributor, commented, "More than 80 young women walked into The National Museum of Computing. I am certain more than one scientist, engineer, and programmer walked out. We saw eyes light up as the realisation dawned upon these young women that technology is as much for them and within their grasp as it is for anyone else."

Career opportunities

Peter Membrey, a software engineer who sponsored the event, said, "I work with many problem-solvers on a daily basis, but sadly there aren't many females among them. I think lots of girls miss out on these career opportunities because currently they are seldom helped to discover that they might have a natural gift for solving problems – which is, after all, what engineering is at its core."

Event organiser Jacqui Garrad is already working on two events for 2019: one exclusively for girls, and another which will encourage teamwork among mixed teams of girls and boys. (HW)

■ A Royal Society report last November drew attention to the scale of the challenge in transforming the way we teach computer science in the UK

RASPBERRY PI IN £78M COMPUTING EDUCATION BOOST

Dedicated funding for Raspberry Pi to transform UK computing education. By Rosie Hattersley

C omputing education in England is about to get a much-needed jolt of funding with the help of the Raspberry Pi Foundation.

The welcome cash injection comes a year after The Royal Society reported that there was a "once-in-a-generation opportunity" to transform the way computing is taught in schools and colleges. Commenting on the

report, Raspberry Pi Foundation CEO Philip Colligan noted that "there's a long way to go before we can say that young people are consistently getting the computing education they need and deserve in UK schools."

The Raspberry Pi Foundation is one of the organisations that, jointly, have secured £78 million in UK government funding to make this vision a reality.

The Foundation is part of a consortium that also includes STEM Learning and the BCS (British Computer Society). Google has also pledged £1 million to support free online computing and computer science courses accessible to anyone.

While existing computing and ICT (information and communications technology) teachers are being directly

■ Teachers will get resources, training, research, and certification as part of the programme

targeted, the scheme will also upskill existing teachers in other disciplines to teach GCSE Computer Science.

Philip explains that the money will be used "to make sure every child in every school in England has access to a world-leading computing education."

The consortium will fund a new National Centre for Computing, with a network of computing hubs, where existing primary and secondary school computing teachers in England will be able to take part in fully funded CPD (continuing professional development) courses. Teachers will also have access to free resources, enabling them to teach computing to students from Key Stage 1 right up to A-Level.

As part of an all-hands-on-deck approach to overhauling computing teaching in England, the Raspberry Pi Foundation and its consortium have more than 60 organisations signed up to offer practical assistance and expertise. Businesses, universities, and non-profit organisations are pooling their expertise and resources to provide the support that educators and schools require.

You can learn more, and get involved, at **teachcomputing.org**. (HW)

EUROPEAN ASTRO PI CHALLENGE PHASE 2

The second phase of Mission Space lab has begun

W e're pleased to tell you that the first phase of Mission Space Lab (of which you can read more about on page 28) was a complete success. Here's a note from the Astro Pi team:

"ESA Education and the Raspberry Pi Foundation are delighted to announce that Phase 2 of the European Astro Pi Challenge: Mission Space Lab has begun. During Phase 1, we received a record-breaking 471 entries from 24 countries! Now, the 378 selected teams will have the chance to write computer programs for the scientific experiments they want to send to the Astro Pi computers aboard the International Space Station (ISS)."

The selected teams will receive an European Astro Pi Challenge kit so that they can develop and test your experiments. The kit includes a Raspberry Pi, two Raspberry Pi Camera Modules, and the Sense HAT. You all have until 6 February to complete your experiments.

A new mission

For those who missed out, we have a new mission for you: Mission Zero. Here's Astro Pi command:

"Young people up to the age of 14 still have to the opportunity to take part in our second challenge, Mission Zero. Teams will have until 20 March 2019 to write a simple program to display their personal message to the astronauts on board. You don't need any special

> **ALL PARTICIPANTS THAT FOLLOW THE GUIDELINES ARE GUARANTEED TO HAVE THEIR PROGRAMS RUN IN SPACE**

equipment or prior coding skills, and all participants that follow the guidelines are guaranteed to have their programs run in space."

Head to the Mission Zero website to learn more: **helloworld.cc/2QkRji5** (HW)

SIGN UP FOR
PICADEMY USA 2019

Free teacher training continues in the US

P icademy is the Raspberry Pi Foundation's free, two-day training programme that helps educators jump start their digital making journey. You'll learn how to bring coding to life and explore new ways to create with technology. Since 2016, Picademy has reached over 790 educators in North America and supported them to teach computing with confidence, creativity, and excitement.

A Picademy event is very special, bringing teachers and educators together to get a better understanding of how to use the Raspberry Pi for computing education. The ones organised by the North American Raspberry Pi Foundation are no different, albeit covering a larger area than the UK.

There were a few recent Picademy events in 2018, including the Seattle one that was held in the amazing Living Computers Museum. "We trained 79 educators hailing from 17 US states and four different countries," says Andrew Collins, Raspberry Pi Educator Training Manager. "All came from a variety of teaching backgrounds: makerspace leaders, teacher trainers, university professors, museum staff, classroom educators."
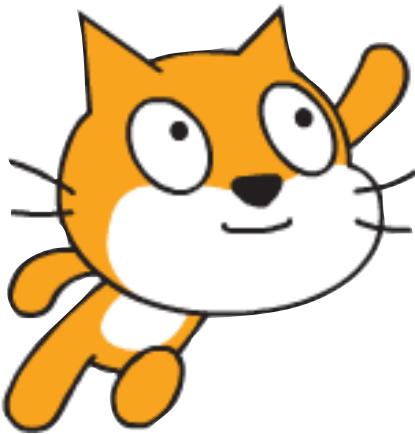
Raspberry Pi will announce the Picademy 2019 season in January. You can sign up to receive an email announcement on the programme once applications are open. Sign up at: **helloworld.cc/2F633Up** (HW)

■ Picademy trainees get parts and equipment to help them learn digital making and computing

■ Picademy graduates from at the Living Computers Museum in Seattle celebrate graduation
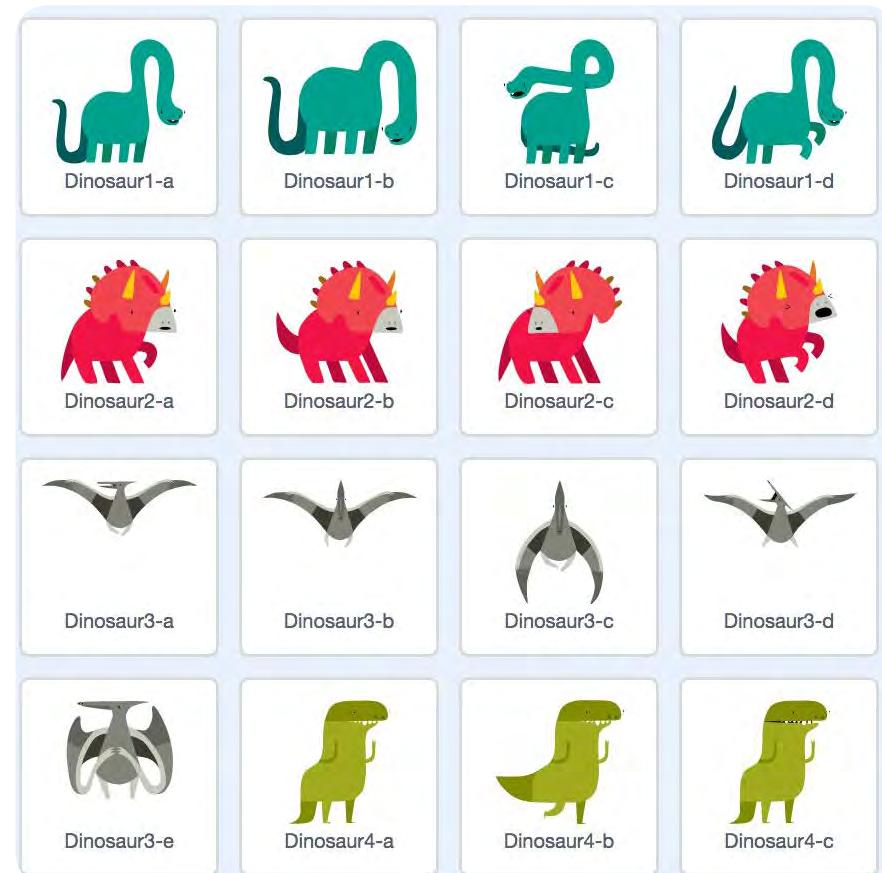
WELCOME TO
SCRATCH 3!

What's changed with Scratch 3 – lower floor, wider walls, same high ceiling

STORY BY Miles Berry

Over the years it has been with us, Scratch has had a profound impact on computing education. Many children's first experience of coding is through Scratch, with the block-based, visual interface making it easy for anyone to get started. The building block-inspired approach encourages playful experimentation, and fosters learning through discovery. The global community of Scratchers provides inspiration, an audience, and encouragement at all stages. Scratch has provided a starting point and inspiration for many other block-based languages, and analysis of the corpus of Scratch projects (or just a subset of it) provides some great insights into children's understanding of computer science.

A new version of Scratch then is something of a milestone for all of us coding or teaching with it, and for the CS education community more generally. As Mitch Resnick highlights in his introduction to this month's cover feature, much has not changed in the upgrade from version 2 to 3, and the Scratch team deserve credit for remaining true to their core principles, prioritising simplicity and accessibility. Computing education pioneer Seymour Papert emphasised the need for a low floor and a high ceiling in technology for learners: it should be easy to get started, but there should be no limit to how far you might take things. Resnick adds to this the idea of wide walls, of many paths up from the floor toward the ceiling. Scratch's new version addresses all three of these elements very effectively.

■ Dinosaur costumes!

Low floor

Scratch 3 makes it easier than ever to get started. There's now a built-in set of tutorials, which you might think of as akin to the build instructions that come with sets of LEGO – you certainly don't have to use these step-by-step guides, but following the plans here gets you started, and many may find this less intimidating than staring at a blank editor window.

The editor has been simplified somewhat. The coding window and the stage have swapped over, bringing the layout back to what we had in version 1.

We've lost a few of the lesser used tools, and the grouped palettes of blocks are now in a single, scrollable list, making it much easier to find the block you're looking for. Some of the less frequently used commands (those for turtle graphics, music, and the webcam) have become extensions – see opposite).

When getting started with teaching (or learning) Scratch, it's easiest to use the built-in library of sprite costumes and backgrounds. It's lovely to see how these have been expanded, with some really high-quality artwork added to provide inspiration to young coders.

Costume and background image editing has been improved, as has the sound editor (check out the cool Echo and Robot effects).

The move from Flash to HTML/JavaScript is very welcome, not least because this makes it much more accessible. Scratch, at last, now works on tablets such as the iPad via the browser – check out Laura Sach's report of piloting this with one school's Code Club. Things like text search and screen

reading work in Scratch 3 if they work in the browser. The language pack can be changed just as easily, and rather wonderfully selecting Arabic and Hebrew swaps the interface and blocks over to suit right to left reading.

Wide walls

Scratch has always supported a wide range of contexts for young people to learn coding: it seems such a shame that in schools many teachers focus on just animations and games, great as they may be. In Scratch 2, a number of extensions were available through ScratchX, but Scratch 3 brings the best of these into core Scratch, as the equivalent of standard libraries. Martin O'Hanlon's article discusses these in detail. Interestingly, some of the functionality that was built into Scratch 1 and 2 now moves into extensions, I think with the aim of making the new version that bit more accessible, while still keeping this functionality present for those who want it. The pen tool for turtle graphics is now one of the extensions, as are the MIDI-like music blocks: I find both of these great for introductory lessons, and so hope that their move into extension land won't mean these great tools get overlooked.

Physical computing is now well supported – ranging from simple input controls with MaKey MaKey and the micro:bit (check out Giles Booth's article on how this works), through to robotics using LEGO's WeDo 2 and Mindstorms EV3 kit. Connectivity via Bluetooth to the LEGO kit is really rather nice – using one family of building blocks to control another family of building blocks.

The extensions also make it possible to use some machine learning tools here: at launch, we have text to speech, which I hope will get Scratchers thinking about how to make their programs a bit more accessible to those who struggle with text on screen, and Google-powered machine translation – we've a quick demo of both presented as a lesson plan further on in this issue. The extension architecture is open to other developers to use with the support of the Scratch team – I've seen demos of speech to text, and I suspect image recognition won't be too far away: see David Horton's experiment with an AI-powered version of

■ LEGO EV3 blocks in Scratch 3

rock, paper, scissors in Scratch for one idea of what we could do with this.

High ceilings

It's not immediately obvious how Scratch 3 extends the ceilings of what young programmers can do. Where Scratch 2 gave us the ability to create our own blocks and pass parameters to them, there's no similar pushing the limits of the CS we can teach with Scratch in this version (I'd have loved to see functions this time round…), but then again, perhaps there doesn't need to be? Scratch is already a Turing complete language, and so any problem that can be solved using a computer can (at least in theory) be solved in Scratch: how high do the ceilings need to be?

There are plenty of languages that folk can move on to from Scratch: I think Snap! is ideal as a second block-based language, and the path from Scratch to Python is oft trodden and well documented. On the other hand, it's worth exploring the Scratch community to see just how far young people have taken Scratch themselves, typically outside of the classroom, using Scratch fluently as a medium for creative expression, rather than just as a means of learning to code – Scratchers' involvement goes way beyond coding, though: it's at its most impressive through participation in the community – liking, commenting, remixing, curating, supporting. Important as computational thinking and coding may be, I suspect it's actually these softer, participatory skills that will serve these amazing young people best in the long term. (HW)

SCRATCH 3

PROJECTS, PASSION, PEERS, AND PLAY

A new version of Scratch is here, and we can't wait to share it!

STORY BY Mitch Resnick
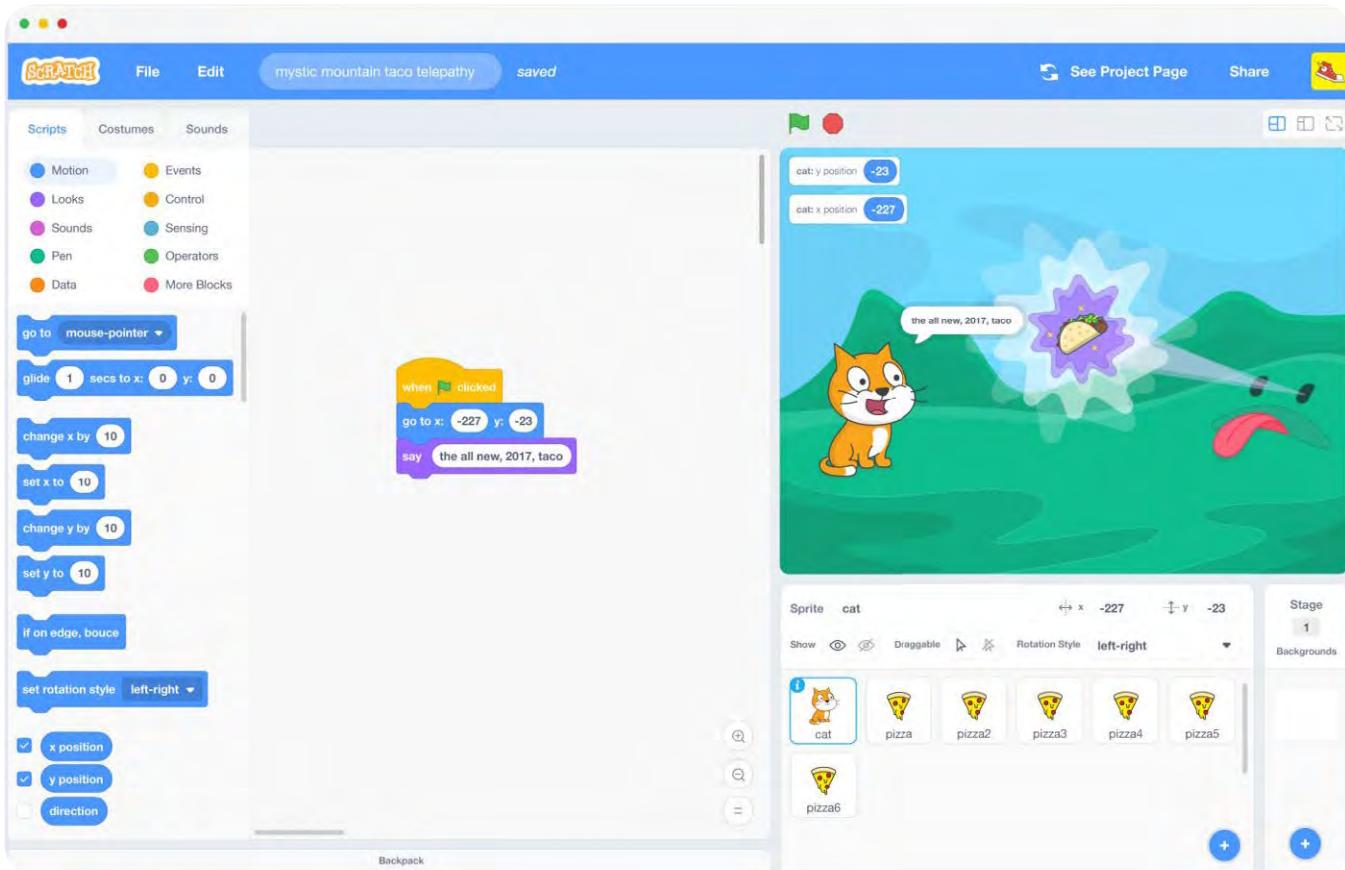
Scratch 3 includes many new things to be excited about. With new 'extensions', you can program Scratch characters to talk out loud, or get inputs from sensors to control your games, stories, and animations. With new video tutorials, it's easier than ever to get started with Scratch – and to go further. And Scratch now runs on a wider range of platforms, including tablets and other touch devices.

But I'm most excited about what's not changing in Scratch 3. The goals and values of Scratch have stayed the same since we launched the first version of Scratch back in May 2007. As we developed new features and capabilities for Scratch 3, we made sure to stay true to our mission of providing all kids, from all backgrounds, with new opportunities to express themselves creatively with new technologies.

How do we stay true to this mission? We're guided by four core principles, which we call the Four P's of Creative Learning: Projects, Passion, Peers, and Play.

Projects

In most introductions to coding, kids are asked to solve puzzles. With Scratch, we focus on projects, not puzzles, so that kids learn the process of turning an initial idea into something that can be shared with others. As they create projects, kids learn not only how to solve problems, but how to find new problems, how to develop new strategies, and how to refine their ideas over time. Kids around the world have shared more than 40 million projects on the Scratch website. We've made sure that all previous projects continue to work in Scratch 3.

Passion

When kids work on things they care about, they're willing to work longer and harder, and persist in the face of difficulties. Since different kids have different passions, it's important that Scratch supports many different types of project. So as we developed new features for Scratch 3 (new extensions, new images, new tutorials),

our goal was to open opportunities for more types of projects, to connect with the interests of more kids.

Peers

Learning involves social interaction, with people sharing ideas, collaborating on projects, and building on one another's work. The Scratch online community supports social interaction in two ways. It provides an

audience: when kids share their projects, they get feedback and suggestions from peers. At the same time, the community provides inspiration: when kids try out projects made by peers, they can get new ideas for their own projects. We designed Scratch 3 to support and encourage more social interaction.

Play

We see play not as an activity, but rather as an attitude: a way of engaging with the world. When people are playful, they are constantly experimenting, trying new things, taking risks, and testing the boundaries. We've designed Scratch to facilitate playful tinkering, encouraging kids to modify code and remix images. In Scratch 3, we focused especially on

> " WE'VE DESIGNED SCRATCH TO FACILITATE PLAYFUL TINKERING, ENCOURAGING KIDS TO MODIFY CODE AND REMIX IMAGES

expanding the ways kids can play with sounds, making it easier for them to record sounds, add special effects, and interact with sounds through new sound effect coding blocks.

We can't wait to see what kids create and share with Scratch 3! **(HW)**

SCRATCH
MEETS MICRO:BIT

Scratch is the lingua franca of learning to code – micro:bit is the lingua franca of physical computing – now they meet!

STORY BY Giles Booth

The more I use Scratch in formal lessons or in Code Clubs, the more I grow to love it – and so do the children who play with it. I use the word 'play' deliberately, for Scratch fosters learning through play, tinkering, and experimentation. It's a rare Scratch session that doesn't end with a child creating something amazing and delightful that neither they, nor their teachers, expected at the start of the lesson.

Created by the Lifelong Kindergarten group at the Massachusetts Institute of Technology's Media Lab in the early 2000s, Scratch has now been translated into dozens of languages and has become an almost universal language of learning to code. It is, I believe, one of the best inventions in the whole field of education, let alone educational technology.

Since its launch by the BBC in 2015, the micro:bit has spread around the world. In October 2016, the Micro:bit Educational Foundation was launched to carry on the project with a mission to inspire every child to create their best digital future through physical computing. From UK secondary schools, over two million micro:bits have now spread worldwide through national schemes in Iceland and Denmark as well as increasing use in education in countries as diverse as Canada, Croatia, Sri Lanka, Singapore, and even in refugee camps in Greece. This astonishing device is uniquely placed to improve learning about technology through physical computing.

Low floor, high ceiling

Like Scratch, the micro:bit has a 'low floor, high ceiling'. In practice, this means something that's easy for children to get

started with, getting very swift results and gratification; it also means they're deceptively simple: amazingly complex things can, and are, made every day with both Scratch and the micro:bit. With a few blocks or lines of computer code, a micro:bit, and a battery pack, children can quickly create autonomous digital artefacts that do entertaining or useful things, whether playing simple physical games like rock, paper scissors, making step counters, or prototype devices to sound an alarm if a tap is left running. University researchers are even pushing the micro:bit to the limit by exploring the possibilities of machine learning in character recognition.

With the new version of Scratch, these two worlds meet, and the possibilities are incredibly exciting. Scratch 3 isn't another editor for micro:bit. It doesn't replace MakeCode or Python, but instead it allows your Scratch projects to interact with the physical world through the micro:bit.

There's no physical connection between the computer and the micro:bit. All interaction happens wirelessly using Bluetooth. Download and flash the Scratch HEX file onto your micro:bit, run the Scratch Link app on your PC or Mac, and add the micro:bit extension blocks then link the micro:bit. Linking uses the individual micro:bit's device name to make sure you're linking the computer with the correct micro:bit – as soon as you flash the Scratch HEX file on to a micro:bit, its name ('getit' in this example) scrolls across the display. Names are based on a unique ID hard-coded into each micro:bit.

Hungry Dino game

This simple game shows how Scratch and the micro:bit can interact. The aim is to guide Dino to chomp tacos and avoid the bugs. You get a point every time you bump into a taco, lose one if you hit a bug.

Scratch can read data from the micro:bit's accelerometer input in real time. Here we sense which way the micro:bit is tilted to control the motion of the dinosaur sprite. Tilt it to the front and the sprite moves up (increasing its position in the Y-axis), tilting back does the opposite. Similarly, tilting left

RESOURCES

There are resources ready for use in class:

- **Five micro:bit Scratch cards featuring stopwatch, dance, drawing, display, and music activities. Download and print the cards in PowerPoint format here: microbit.org/scratch**

- **You'll also find some starter projects on the Scratch website: scratch.mit.edu/microbit**

- **Make a spooky Bat Theremin musical instrument: helloworld.cc/2RqnelR**

and right changes the sprite's position in the X-axis. With just a few blocks, children can make their own wireless physical game controller like the real ones they'll probably be very familiar with indeed!

The micro:bit can also display information from Scratch: another 'forever' block keeps the micro:bit's display output updated with the current score, useful if you're a long way from the screen or want to provide another kind of visual feedback.

Taking it further

In addition to the blocks used in this game, Scratch can draw your own pictures on the micro:bit display, it can trigger events when the micro:bit is moved, shaken, or jumps, opening up new worlds of gesture control. The angle of tilt can be measured, which could be used in science experiments or for more precise control. Scratch can also access the three touch pin inputs on the micro:bit, opening up access from simple electronic circuits which could be made

with tin foil, copper tape, or anything that conducts electricity.

Children are the best inventors, and I can't wait to see what they create with micro:bit and Scratch together! (HW)

Giles is the Educational Content Manager for the Micro:bit Educational Foundation. He is a Raspberry Pi Certified Educator and has taught Computer Science in both primary and secondary schools in London.

SCRATCH

NEW EXTENSIONS

Version 3 of Scratch features some new extensions that allow you to do some fun and exciting things that 'normal' Scratch can't do

One of the most exciting developments in Scratch 3 is its new extension platform.
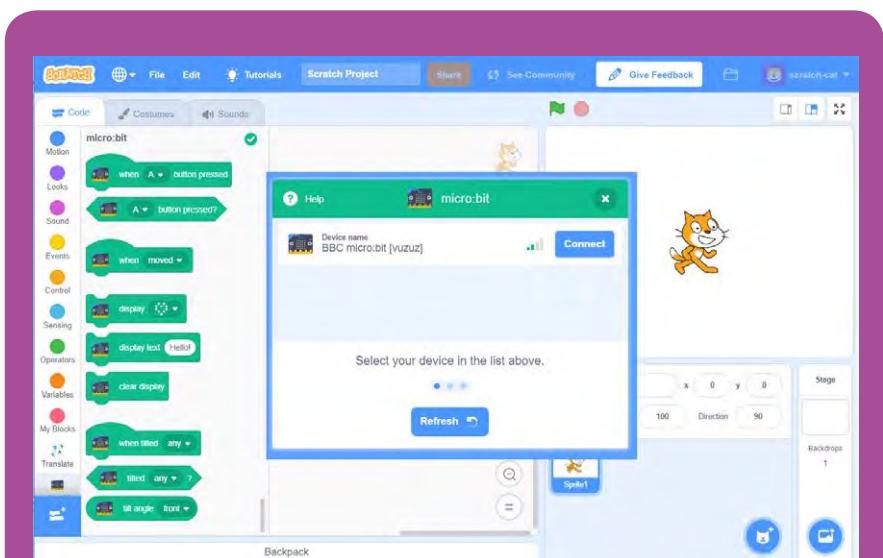
As the name 'extension' suggests, this extends the functionality of Scratch, allowing you to do exciting things that the 'normal' Scratch can't do, such as translate text or drive robots.

In previous versions of Scratch, extensions were experimental or unsupported, the website **scratchx.org** being the main repository. And while there were many which worked well, were well maintained, and had good documentation, there was an equal number which didn't work or provided no support to get started.

Scratch 3 changes this by making extensions part of the system, with processes for developers to get their extensions into Scratch. Such a radical change does, however, mean that old Scratch 2 extensions aren't compatible and won't be available in Scratch 3.

Extensions are core in Scratch 3

At the time of launch, it's expected that there will be eight extensions, allowing you to create music, draw with sprites, sense motion with a camera, turn text into speech,

USING SCRATCH 3 WITH A MICRO:BIT

Controlling a micro:bit with Scratch 3 is simple:

■ Visit scratch.mit.edu/microbit
■ Download and install Scratch Link
■ Download the micro:bit hex file and copy to your micro:bit
■ Add the micro:bit extension in Scratch 3
■ Connect to your micro:bit and click 'Goto Editor'

■ Click the "display text 'Hello'" block to see the message on your micro:bit

Once connected, you can use the micro:bit blocks in Scratch, perhaps use the buttons to control a sprite, or display a game's highscore on the LED matrix.
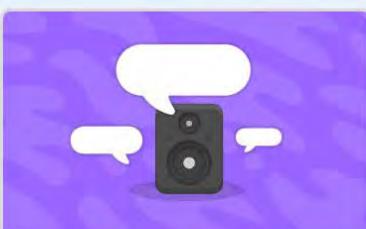
Music
Play instruments and drums.

Pen
Draw with your sprites.
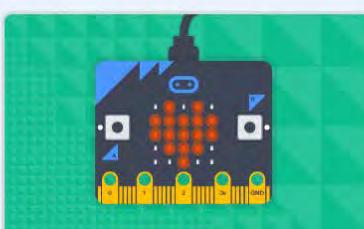
Video Sensing
Sense motion with the camera.
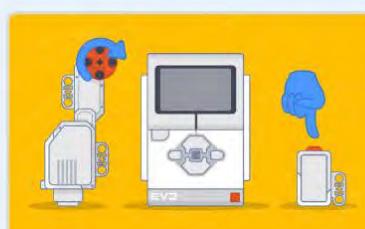
Text to Speech
Make your projects talk.
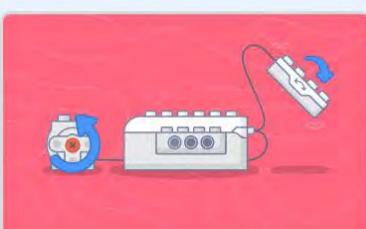
Translate
Translate text into many languages.

micro:bit
Connect your projects with the world.
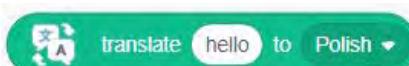
LEGO MINDSTORMS EV3
Build interactive robots and more.

LEGO WeDo 2.0
Build with motors and sensors.

■ There will be eight extensions in Scratch 3 when it launches, allowing you to translate text into different languages, play music, or connect to LEGO robots

translate text, connect to a micro:bit and LEGO Mindstorms, and WeDo.

Adding an extension in Scratch 3 is simple: click on the 'Add Extension' button in the bottom left of the screen and select the extension you want. Doing so will bring in a new set of mint green blocks which you can attach to your programs just like

The video sensing extension utilises the webcam in your computer to detect motion and also the direction of the motion, allowing you to control your Scratch projects by moving in front of the camera.

You can use 'text to speech' to replace speech bubbles for 'say' blocks with a synthesised voice. There are a variety of different voices (including squeak and kitten); combine this with the translation

a computer which runs either Windows or macOS and can use Bluetooth. There are some really helpful setup instructions on the micro:bit extension help page at **scratch.mit.edu/microbit**.

micro:bit and LEGO extensions need Scratch Link

The new extensions in Scratch 3 present alternative options for engaging your learners. For those students who are pushing the boundaries of what you can do with Scratch, they'll welcome the new challenges and opportunities to learn new concepts. Students who aren't engaged by the typical Scratch projects, such as games and graphics, may be motivated by the alternative options of physical computing, music, and drawing.

> " **THE NEW EXTENSIONS IN SCRATCH 3 PRESENT ALTERNATIVE OPTIONS FOR ENGAGING YOUR LEARNER**

any other block, such as the translate "'hello' to 'Polish'" block.

Two of the new extensions, Music and Pen, previously part of the main Scratch blocks, have moved into extensions to reduce the number of code blocks available and simplify the user interface. While this introduces a small amount of friction to getting started, the benefits are obvious.

extension blocks, and you have the ability to get the Scratch cat speaking kitten in Polish.

To use the micro:bit and LEGO extensions, you'll need to install 'Scratch Link', a small piece of software which allows Scratch to talk to the hardware. To use Scratch Link, and therefore the micro:bit and LEGO extensions, you'll need

Give your students the opportunity to experiment with the extensions, understand their capabilities, and use them to change an existing project using the micro:bit, LEGO Mindstorms, and WeDo extensions to take Scratch out of the computer screen and into the real world. (HW)

USING SCRATCH 3
ON A TABLET

Scratch 3 was on trial for a while before it landed, but what did a school's Code Club in Gloucestershire make of it?

STORY BY Laura Sach

A rguably one of the most eagerly awaited features of Scratch 3 is the ability to use the much-loved programming tool on tablets alongside the already popular option for pre-readers, Scratch Jr. Many schools already have access to a bank of tablets, and this will allow more children to use existing hardware in school to help them learn to code.

Children in Years 5 and 6 at Andoversford Primary School in Gloucestershire attend a weekly Code Club. These students have been cutting-edge pioneers helping the Raspberry Pi Foundation to test out how well the Code Club projects work using the Scratch 3 beta on iPads.

The students have been working on the projects in the first Code Club Scratch module, and have enjoyed creating a virtual rock band, creating a ghost catching game, and programming their own personal chatbot. Having already experienced working on the school laptops with Scratch 2 for some of the projects, it was an interesting comparison for them to

Head Teacher Rachel Bradley-McKay, who runs the Code Club, said "We decided to trial the beta version of Scratch 3 as our iPads are often more reliable than the laptops in school – plus the children were thrilled to be involved at this test

> ## " THE BETA VERSION OF SCRATCH 3.0 SEEMED OVERALL TO BE A DIRECT IMPROVEMENT UPON ITS PREDECESSOR

attempt subsequent projects on the iPad mini, both from the point of view of using the new Scratch 3 interface and the way that it works on touchscreen hardware.

stage. As someone without a background in tech, I found the interface much easier to use, and found myself coding alongside the pupils."

The students had absolutely no trouble at all finding the code blocks or functions they needed in Scratch 3, even when some of them have been moved or look slightly different to their Scratch 2 counterparts. Head Teacher Rachel commented: "I have been really impressed with how the children have been able to develop their problem-solving skills, adapting project instructions to suit the new platform." Perhaps one of the largest changes has been moving the stage to the right-hand side of the screen, based on research by the MIT Scratch team. The 'pen' and 'music' tools are also now separate extensions that must be specifically added, rather than being core Scratch functionality, with the idea being to simplify getting started for beginners.

Dragging, dropping, and tapping blocks were all frustration free, and the interface has been designed to be tablet friendly, so the blocks are slightly larger and the text is easier to read than in Scratch 2. Some students had an occasional issue with accidentally pinch zooming in too far on the screen, but this can be rectified by pinching out from a point outside of the stage, and seems likely to be less of a problem on a full-sized iPad. Using the full-screen mode with a project that involves keyboard input is a little tricky, as the onscreen keyboard causes the stage window to resize. However, the students felt very positive and enthusiastic, and really took the majority of the changes in their stride:

■ "I think that Scratch 3 is a lot easier on the iPads and it is a lot quicker" – Jake

■ "The new scratch has improved a lot compared to Scratch 2 – I really enjoyed testing it. I also found it much easier to use. I think Scratch 3 is amazing!" – Chris

■ "Scratch 3 on the iPads is so much easier – the sprites are infinitely better and it is much more advanced, while being easier to use" – Elliott

The beta version of Scratch 3 that the students tested seemed overall to be a direct improvement upon its predecessor. The biggest drawback at present is that 'key pressed' blocks and all 'right-click' functionality, such as duplicating blocks, can't be used on a tablet yet, but MIT reports that this should be available later in 2019. Extensions that make use of hardware such as the micro:bit or LEGO Mindstorms are also not available on tablets. However, in the meantime, there are plenty of new features to explore, including a text to speech extension and translation capabilities, and these can easily be taken advantage of with the multimedia capabilities of an iPad. (HW)

SCRATCH CONFERENCE EUROPE DETAILS

The Raspberry Pi Foundation is excited to be hosting Scratch Conference Europe on 23-25 August 2019 at Churchill College, Cambridge, UK.
With talks, informal meet-ups, and plenty of interactive workshops, Scratch Conference Europe offers hundreds of educators an opportunity to explore the creative ways that people are programming and learning with Scratch. Join us to become part of a growing community, discover how the Raspberry Pi Foundation can support you further, and develop your skills with Scratch as a creative tool for learning.

The call for content proposals from the community will open Friday 18 January, and tickets go on sale in April. Subscribe to Raspberry Pi LEARN, our monthly newsletter for educators, or keep an eye on **@Raspberry_Pi** on Twitter for updates if you would like to contribute or attend!

#INSIGHTS

HOW CHILDREN MAKE DIGITAL PROJECTS
RESEARCH FROM COOLEST PROJECTS 2018

STORY BY Lucia Flóriánová

Coolest Projects is a 'science fair'-style exhibition that takes place all over the world. Young digital makers bring their projects to share with others. Judges award some projects in each category, but the main emphasis is on sharing and learning from others.

This year, the Raspberry Pi Foundation research team conducted research at the International and UK events. We explored how children create with technology and what makes the events successful and special. Most importantly, we found what makes children excited while learning, and motivates them to take projects and learning to the next level.

Making digital projects
Problems and ideas, technology and skills
Projects had different stories. Some children started with identifying a problem that they wanted to solve and then thought of ways to achieve it. Others started with looking at their skills or technology they had available and then found ways to use those in a project.

All children had to balance the relationship between three areas:
1) Compelling ideas or social problems they wanted to solve
2) Technology they had available
3) Skills they had or needed to develop to complete a project.

Adult mentors helped them negotiate the balance and refine the ideas so that they were realistic and achievable.

Teams and roles

Some projects were created by individuals and some by teams. In teams, participants either collaborated flexibly or took on specific roles. Sometimes the roles related to different technical aspects, such as a 'Python Programmer' or a 'Hardware Engineer'. In other teams, some children did the code and others took on non-technical roles, such as writing a story or music for a game, or developed artwork.

Some children are deeply immersed in the project creation, others take roles such as 'Games Tester', or just join in to see what it's all about. Through attending alongside their friends and undertaking 'peripheral tasks', newcomers become familiar with practices and language within the community. It can be a beginning of their own journey of becoming practitioners themselves. Lave & Wenger call this "legitimate peripheral participation".

Why Coolest Projects are so great

Agenda, drive, and learning

Children were very invested in projects they brought to the event and what they learned. Many said they made more ambitious projects than they would normally do or took their existing ones to the next level.

This was because they had set their sights on the event and wanted to create something special. Most children spent a long time on their projects and were very passionate about them. They often had a desired result in mind and worked hard to achieve it. This required learning something new, such as programming concepts or languages. Children often mentioned that they "didn't know how to do that" or "how to fix this problem", but then intentionally learned it. Learning became a part of fun and agenda – children learned with enthusiasm because they decided they needed to, in order to achieve the result they desired. It was also made more effective because it happened in a context and children got to immediately apply what they had learned in practice.

However, it's important for children to have an adult or materials that guide them and help them troubleshoot when they get stuck. It's also important that children don't set unachievable goals that result in frustration.

Combining programming with other domains

Many projects were interdisciplinary and combined digital making with knowledge

IN THE CLASSROOM

Programming in schools turned out to be an important influence on many children at the events. A number of children said their first steps in programming were taken in primary school. It was through these early positive experiences that they became interested in digital making.

There are simple ways you can use what we found worked in the classroom. For example, sharing projects with each other at the end of a lesson or looking for interesting examples online is one way of getting inspiration. Motivating children to think about goals or benefits their project will have on community can create a similar investment in learning. Presenting projects at a small exhibition for parents, school open days or an assembly can help children feel important and build confidence.

in other areas, such as health, security, environment, or robotics. Children often created projects that used technology to solve a real-world problem that they noticed and cared about. Such an approach to computing can trigger ▶

> interest in technology and motivate children with different interests to get involved with digital making. It can help them see that technology can be used as a tool for achieving a particular goal, and isn't necessarily the goal itself. Projects also became a medium for interdisciplinary learning. Digital making can support other subjects beyond computing, bringing powerful tools to these areas.

Sharing and confidence

Even children who seemed initially shy presented their projects with confidence. Children deeply understood the projects, and focused on explaining how projects work and how they made it. This took away the awkwardness of talking to strange and diverse audiences. They benefited from talking about something that they were passionate about, and seeing other people's interest. Through speaking about something that they know intimately and feel proud of, children experience success. They can gain confidence and become more comfortable with presenting in general.

Inspiration and power of seeing other projects

Children often look for inspiration in projects done by others and examples of what they could do with technology and skills they have. Coolest Projects enables children to do just that; to look at projects at their own level and see what they could do. Seeing projects of other participants was children's favourite part of both events. They spent a lot of time walking around the exhibition and asking other participants about their work. Some children said they felt more confident after seeing that their projects are similarly or more advanced than others people's. Others found an inspiration for improvements they could make to their own projects. They were impressed by the potential that an idea similar to theirs can have.

You can read a full report and other research we have done at: **rpf.io/research**

You can find out more about taking part in Coolest Projects 2019 in Ireland, the UK and North America here: **coolestprojects.org** (HW)

SCRATCH AT ALL LEVELS

Many of the projects we've seen were programmed in Scratch. They varied from full beginners projects to very elaborate ones; and we were delighted to see that children with all levels of experience benefited from showcasing their work.

In **Pollution game (or Shadow Neighbours)**, two boys aged seven and eight created a simple Scratch game that served to raise awareness about pollution and its negative effects. It had a crab character catching plastic waste and was controlled by keyboard.

The Rebel Quiz was a project programmed by two girls aged eight. They wanted to provide inspiring role models for girls by introducing strong female characters through the quiz. After a player answered a series of questions about their personality and interests, they were told which 'Rebel Girl' they were.

Dance Magic was a Scratch game with an element of physical computing. The simple dance game told a player where they should move. The player then had to press the right pressure pad with a right, left, up or down arrow. Girls were inspired by a famous dance game.

The Random Games was an elaborate project done by a 16-year-old. He combined six different games of diverse genres in one computer program. Some of the games included a player 'playing a piano' or a character controlled on a 3D interface. This refutes the assumption that Scratch is only for beginners and younger children.

CAS AS A COMMUNITY OF PRACTICE

STORY BY Jonathan Dickins

Effective delivery of the computing curriculum in UK schools relies upon well-trained teachers, but for some former ICT teachers this has represented an enormous challenge. A new paper in the *Journal of Computing Education* by Sue Sentance and Simon Humphreys explores how situated learning theory can explain how professional development in Computing at School (CAS) works effectively.

Situated learning theory holds that learning takes place within communities of practice, and that learning must be situated in a teacher's experience. There are three important concepts within situated learning:

Communities of practice

A community of practice is a group of people working in the same domain with the same set of goals, sharing their knowledge and experiences. CAS has evolved as a community of those interested in computing education, with sharing of resources, ideas, and expertise. They can be said to be operating within multiple communities of practice, through their website, which includes a forum and resource sharing, and their Regional Centres and local hubs.

Legitimate peripheral participation

Legitimate peripheral participation is the idea of being a newcomer, observing the culture of practice, but not yet being a full part of the community. There are a number of routes by which a member of a community can go from peripheral to full participation. CAS Master Teachers are experienced teachers who work with others in their local community, and many teachers who were new to computing three or four years ago are now Master

> ❝ PARTICIPATION BENEFITS EVERYONE; THE NEWCOMER SHARES KNOWLEDGE AND GAINS CONFIDENCE IN THEIR TEACHING

Teachers. The BCS Certificate in Computer Science Education enables teachers to develop a project in their classrooms over the course of a year, which increases their confidence and may lead to more central participation. These are examples of the way that teachers of computing are moving into more central roles in the community of practice to which they belong.

Continuity displacement

Continuity displacement is an iterative process in which the community evolves via the change that newcomers bring.

A first step for a newcomer to CAS could be to join the online community, becoming familiar with the topics under discussion.

They might go on to attend a CAS hub meeting, sharing their experiences and perhaps also discussing them in the online community. Participation benefits everyone; the newcomer shares knowledge and skills valuable to others, and gains confidence in their teaching.

Analysing CAS in this way, the authors conclude that the CAS community must continue to change as newcomers become more central participants, and that understanding how to increase participation can inform intervention and enable more community-focused professional learning for computing teachers.

You can read the full journal article here: **helloworld.cc/2Gzr4ES** (HW)

Credit: ESA

■ ESA Astronaut Tim Peake with the Astro Pi computer on board the International Space Station

MISSION ZERO
YOUR PROGRAM IN SPACE!

Astro Pi is a great way to use the excitement of space to engage
young people in science, computing and digital making

STORY BY Erin Brindley

The European Astro Pi Challenge is an ESA Education project run in collaboration with the Raspberry Pi Foundation. It offers students and young people the amazing opportunity to conduct scientific investigations in space, by writing computer programs that run on Raspberry Pi computers on board the International Space Station (ISS). The Astro Pi challenge is divided into two separate missions featuring different levels of complexity: Mission Zero and Mission Space Lab. Mission Space Lab, the

advanced mission, closed for entries on 1 November, and this year saw 471 entries from students across Europe and Canada, who had designed experiments to be run in space.

Mission Zero

Mission Zero, the beginner mission, opened on 29 October and has already seen over 1000 teams take part. It runs until 20 March and offers students up to 14 years old the chance to have their program run on the ISS. Teams write a simple program

to display a message and the temperature readings on the Astro Pi computer, available for the astronauts to see as they go about their daily tasks. The challenge is completely free, and open to students and young people from all ESA member and associate member states (see **astro-pi.org** for more details).

The activity is designed to be flexible to differing levels of familiarity with Python programming. With the support of the online resource, the challenge can be completed by complete beginners and

can be a useful way to facilitate the jump from block- to text-based programming. Additionally, more challenge can be added to the task by coding pictures to be displayed or changing the colour and speed of the scrolling text.

Darren Bayliss, Code Club Regional Coordinator, had this to say about his experience running Mission Zero in a Code Club: "I ran it this morning with my second group of kids and they absolutely loved it, and whilst some found it a challenge they still can't comprehend that they wrote code that will actually be run on the ISS."

All teams who submit an eligible entry will also receive a certificate of completion, which shows the exact date, time, and location of the ISS at the time their program was run.

Taking part

Mission Zero can be completed in an afternoon and on any computer with internet access. Students and young people work in teams of two to four people, and follow along with our online resource to write a short Python program that shows their chosen message for the ISS astronauts and an air temperature reading on the Astro Pi LED screen. No extra hardware is needed, and everything can be done in a web browser. The program is written in our online Trinket emulator, which allows students to see a simulation of their code running on the Astro Pi before they submit it.

Mission Zero is designed to be easy to complete in a classroom, Code Club, CoderDojo, or Raspberry Jam. The teacher or mentor registers on the Trinket website for a classroom code. Students then use

MISSION ZERO IN A NUTSHELL

Credit: ESA

Mission Zero offers students and young people the chance to have their computer programs run in space on the International Space Station!

HOW TO TAKE PART:
1. Teachers and mentors register for a classroom code
2. Students and young people follow online resources and submit their entry
3. All entries submitted are guaranteed to run in space!

WHO CAN TAKE PART?
- 14 years or younger
- Teams of two to four
- Supervised by a teacher or mentor
- From an ESA member/associate member state

Open until 20 March 2019

> ## THE ACTIVITY IS DESIGNED TO BE FLEXIBLE TO DIFFERING LEVELS OF FAMILIARITY WITH PYTHON PROGRAMMING

the online resource or the printable version to walk through the activity. The online emulator automatically checks the program to ensure that it adheres to programme rules. Students can then submit their code using their mentor's classroom code.

This year, Mission Zero has been expanded into all 20 official ESA languages, with the hope of making the programme more accessible to students and young people in non-English speaking countries too.

Mission Zero is an amazing way for young people to get excited about computing through the medium of space travel. It helps students understand the connection between learning these skills and the interesting ways they can be utilised in their future careers. The activity is easy, fun, and free! Take part today by heading to **astro-pi.org.** (HW)

Erin is Programme Manager for the Raspberry Pi Foundation.

COMPUTING FOR STUDENTS
WITH SPECIAL EDUCATIONAL NEEDS AND DISABILITIES

How can we engage and include all students in computing? The results of a national survey into the state of computing in special needs settings may benefit teachers and students in all sectors

STORY BY Catherine Elliott

S ince the computing curriculum was announced, I've had the pleasure of working with staff and students in a range of special schools to look at how to adapt the programme of study for young people with special educational needs and disabilities (SEND). With over a million students with special needs in mainstream schools, the activities and strategies developed in special schools could be used effectively in every classroom to ensure we include all young people in computing.

In autumn of 2017, I invited responses to a survey from computing teachers in special needs settings across England, in order to establish the current picture of the subject, and investigate the strategies and resources that are working well for their SEND learners.

Results of the survey

The survey was completed by teachers representing approximately 6% of special schools in England, whose students have a wide range of special needs and disabilities. The results indicated a similar picture to mainstream schools, with teachers generally feeling confident in teaching the computing curriculum, but with some concerns about the newer computer science aspects. The barriers to teaching the subject included issues with the technology and lack of staff confidence. Overwhelmingly, however, the main barrier mentioned was the lack of SEND-specific materials and resources.

■ Unplugged activities can provide a physical representation of an abstract concept using a familiar context to support understanding

Two-thirds of the teachers surveyed considered the subject of computing to be relevant to their learners; as a key life skill, a way of accessing the wider curriculum, and as an engaging subject many students can excel in: "Many of our students are very engaged by ICT. It empowers students who struggle with social interaction to present and share their work widely." Only a third of the respondents, however, felt that the computer science elements of the computing curriculum were relevant. For many learners, it was considered irrelevant

due to their low literacy and cognitive levels. Where it was considered meaningful, teachers cited a number of examples where programming and computational thinking activities contributed to learning about other subjects. Computational thinking also seems to offer a useful framework for solving problems in other areas, where the language and concepts are taught explicitly across subjects: "We did a whole topic before the summer based on algorithms [...] The pupils then took that into maths and started to create their own algorithms for doing angles,

for example, in triangles. They created their own algorithms, sequenced them, passed them to another pupil, debugged them then took it back and amended it. And it really did help in maths."

Effective strategies and resources

In terms of the resources that are being used, the most frequently cited are very similar to in any mainstream school. A great deal of work is then happening to adapt existing resources for the specific needs of the students. For example, in a school for visually impaired students, the teacher has written some code to make a programming environment for the BBC micro:bit that works with a screen reader. In a school with a high proportion of students with profound and multiple learning difficulties, they use special switches so that their young people can control everyday technology with a single push of a button, and start to explore simple cause and effect.

about programming by nearly a third of the teachers. By far the most popular was the Bee-Bot or Blue-Bot, the floor robot from TTS. Newer alternatives mentioned were the Code-A-Pillar and Cubetto. Both of these provide a physical representation of the sequence of a program, created by snapping together body parts or adding command tiles in order, which can help students to make the connection between their program and the movement of the robot.

Other physical computing devices, such as the LEGO Mindstorms robots, Crumble Controller, and BBC micro:bit are being used successfully to introduce more complex programming. The multi-modality of such devices plays a large part in their success with students with SEND, where outputs can include lights, sound, and movement, ensuring that even those with visual or hearing impairments can be included. The immediacy of the output is also motivating as a very short program can quickly yield a

> ❝ TWO-THIRDS OF TEACHERS SURVEYED CONSIDERED THE SUBJECT OF COMPUTING TO BE RELEVANT TO THEIR LEARNERS

Guidance from Jisc for teachers of students with learning difficulties includes the following rules:

- Make learning participative
- Encourage peer learning
- Break down tasks into smaller steps that will incrementally build into the task objective
- Use learners' own words, language, materials, and personal context (**www.jisc.ac.uk**, 2014)

These are easily incorporated into the computing classroom, in particular using unplugged activities and physical computing devices, both of which were the most frequently mentioned effective approaches to teaching programming and computational thinking in the survey.

Hands on and unplugged

Physical computing devices were mentioned as being successful in teaching students

tangible output. Seymour Papert referred to the original floor turtles as "objects-to-think-with" (Papert, 1980), and the physical connection with the program appears to help with understanding for many learners.

The other popular strategy for teaching computing concepts in the survey was the use of unplugged activities. As Caldwell and Smith note in their book *Teaching Computing Unplugged in Primary Schools*, technology can be distracting for pupils learning new concepts, and "what can be much more useful for learners is to step away from the technology and practise the computational thinking skills in a different setting. If the physical trappings of the new setting are simple and well-known, such as pencil and paper, beanbags and hoops, we don't need to devote any energy to making them work and can instead focus on the new skills and knowledge we are developing." (Caldwell & Smith, 2017) Unplugged activities also provide a way of making the abstract more concrete, "providing a physical

representation, the learner can point to" (Curzon et al. in Computer Science Education, ed. Sentence et al, 2018), allowing students to walk through or manipulate an idea, rather than grapple with it in their heads.

This survey is a starting point to gauge the current picture of computing in special schools. The hope is that further research can be conducted into effective strategies for learning that will benefit students in every classroom. For the full report, please visit **helloworld.cc/2UYRe7b** (HW)

Catherine is an eLearning Consultant with the Sheffield City Council eLearning Service. She has spent the last five years looking at how to adapt the computing curriculum for learners with SEND. She is joint leader of the Sheffield & South Yorkshire Secondary CAS hub.

SUBSCRIBE
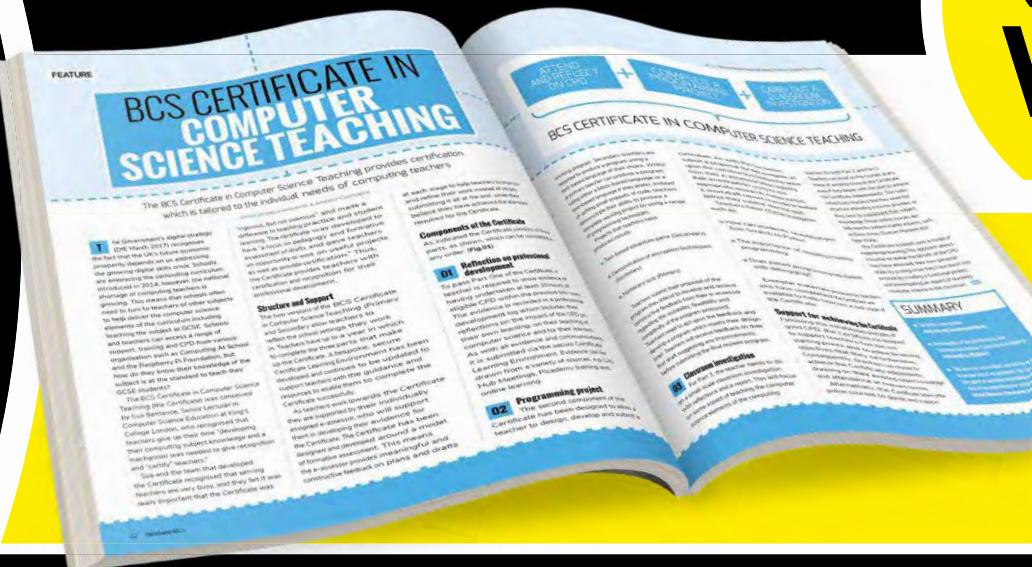
Sign up today for a year - prices start at FREE!

THE MAGAZINE FOR COMPUTING
& DIGITAL MAKING EDUCATORS

(Hello World)

THE NATIONAL CENTRE
FOR COMPUTING
EDUCATION
What is it, and how can it help?

Issue 7 | Spring Term 2019 helloworld.cc

SCRATCH 3

- Introduction to the new Scratch
- micro:bits and tablets
- New extensions
- And much more

GET YOUR
PROGRAM
INTO SPACE
Take the European
Astro Pi Challenge

WHAT'S THE BEST
WAY TO TEACH
PROGRAMMING?

SPECIAL NEEDS
National survey
findings explored

COOLEST PROJECTS 2018

PLUS MAKING A 3D ANIMATED FILM • WHY COMPUTING HISTORY MATTERS • DIETING WITH GENETIC ALGORITHMS
RESEARCH • CLEVER CAT MACHINE LEARNING • MICRO:BIT AND MR BIT

SUBSCRIBE TODAY

- Get three term-time issues
- Have them delivered directly to your door
- Hello World is not available in stores!

LETTING STUDENTS MAKE THEIR OWN 3D ANIMATED FILM

3D animation still requires a hefty computer to bring it to the classroom.
But maybe something more intensive may be of use…
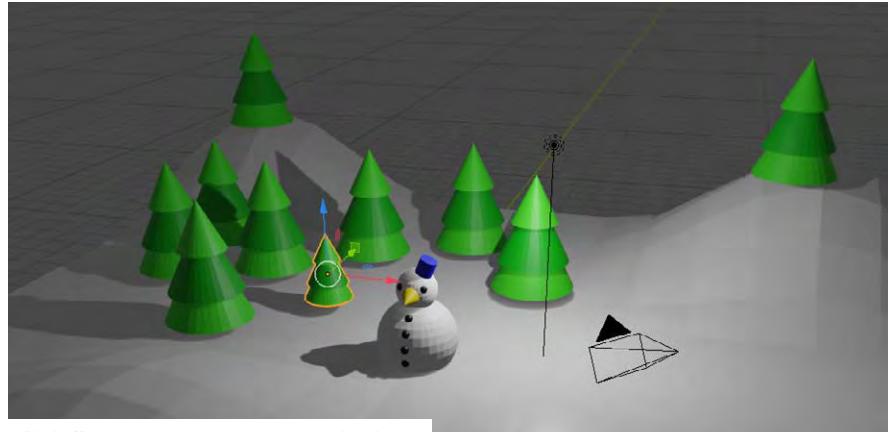
STORY BY Peter Kemp

One of the reasons given for the inclusion of programming in the English school computing curriculum was the idea of democratising digital technology: we wanted people to be creators of the systems that they consume. Apps, websites, computer programs, and games are built using programming, therefore we need to teach our children how to code.

However, coding is not the only component of digital creativity. Much of what you see on the film screen, TV, and in computer games has a 3D digital component; we are also seeing the emergence of virtual and augmented reality technologies. If we want to democratise digital culture, we also need to enable people to create 3D digital content. But how can we go about this?

ICT has been replaced in schools by computing, which has more of a focus on computer science. Where ICT courses might previously have had digital art components, the replacement qualifications, in computer science, don't. Even if a school wanted to teach 3D digital animation, maybe through its media studies course, this would require teachers with the skills to teach it, potentially expensive software, and pretty hefty computer hardware. 3D animation remains one of the last areas of digital creativity that requires some serious computing power to get involved. You can program industry-standard Python through a web browser, and most programs a

■ Blender 3D computer graphics software is free and used in industry

beginner will write will complete almost instantaneously, but to create with 3D animation, you're going to need a fairly recent computer and a dedicated graphics card.
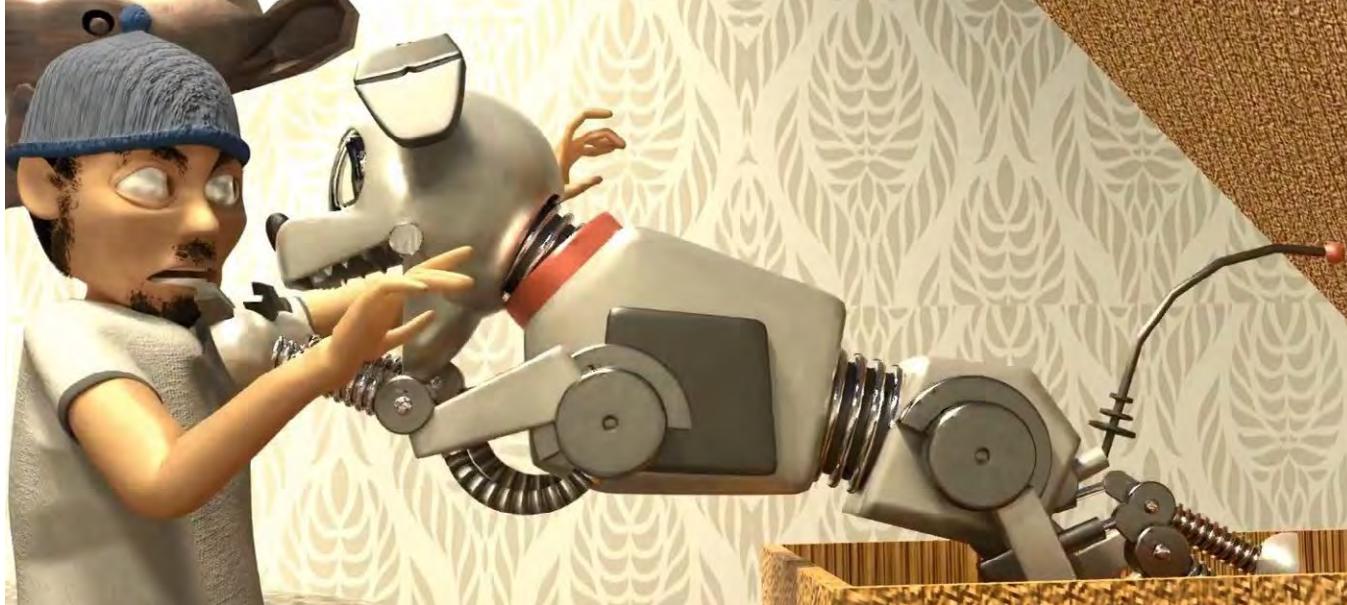
Even then, if you make short films, you'll need to wait hours, if not days, for something to 'render' out – that is, to make each of the individual frames of a film. Once the artists had finished making the characters, props, and shots, Pixar's *Monster's University* would have taken 495.78 years to 'render' on a single computer (the company used thousands of computers to get around this problem, something obviously not available to schoolchildren).

Foundations

We founded 3Dami in 2012 with the intention of giving students the tools, skills,

and contacts to start making their own 3D digital content. I was a secondary school teacher at the time with lots of students interested in getting into film, games, and animation. There was, and still is, a shortage of people working in these areas, with the recent Nesta report *Which digital skills do you really need?* listing animation and multimedia production as the top areas of growth for jobs in the digital sector.

Film, games, and animation are quite straightforward careers to get children interested in, as they are often already avid consumers, but it's less straightforward to get students making informed decisions about choosing these careers based on actual experience. We wanted students to gain real experiences using industry-standard tools, so they could better pick future education and careers.

A 3Dami camp runs for seven days, where teams of nine students create every part of an animated short. On the first day, the students write a story for their team to work to – this means lots of paper and pens, and a complete, albeit paper-based, storyboard by the end of the day. Students then work on computers to create all the props, characters, sets, animations, and shots that make their film, with the première occurring on day seven. Amongst the nine students, two are given roles as director and producer, with the director taking an artistic lead, and the producer using an asset management system to allocate tasks to the rest of the team and queue up the shots for rendering. We set up a 'render farm' – hundreds of computers linked together to speed things up. A shot of seven seconds at 24 frames a second is 168 separate

such as modular design and computational efficiency are natural to 3D animation. One student might make a character that appears in multiple shots, another might make a bucket used in some of the same shots. Once these assets are linked together, we have a film, and if there's an artistic change to the character's hair, because it's linked, it will be immediately updated in all of the shots. Learning the modular design through programming is much more difficult – a buggy module will likely break a program, a buggy haircut will just look a little rubbish. Trying to get shots rendered in time is always a problem, and students need to think carefully about how they set up their animations. Inefficient shots can bring even the fastest computer to its knees for hours at a time; in particular, smoke simulations and hair can be very

■ Visit the Raspberry Pi website for projects to get started

effects in the TV series *The Man in the High Castle*. Blender runs on old hardware that you often find in a school. If you can't get permission to install it, it's less than a gigabyte and will run off a USB stick or shared drive.

Many of our students are now studying digital art, film, computer science, and engineering at university, with several of them now working in the games and film industries. 3D digital art is a shortage area which many students love. If you'd like to get started, please check out the materials we've made for schoolchildren over at **b3d101.org** or on the Raspberry Pi website: **helloworld.cc/2Bv5ZFZ** (HW)

" IT'S IMPORTANT FOR STUDENTS TO USE INDUSTRY TOOLS TO GET A TRUE FEEL FOR WHAT IT'S LIKE TO WORK IN AN AREA

frames, so a normal computer might take 45 minutes to do one frame, therefore 5.25 days to complete the shot. If we split the 168 across 168 computers, the shot would be ready in 45 minutes.

Ownership
The aim is for students to own their film and the management of it. We don't use pre-made assets and avoid lecturing. Students tend to learn from each other and only ask questions when people on their team can't help. Computing concepts

compute intensive. Very rarely does someone learning how to program meet a problem where they need to write more efficient code; thinking about efficiency is a daily occurrence in 3D animation.

We feel it's important for students to use industry tools where possible, as this is the only way that students can get a true feel for what it's like to work in an area. We base our work around the open source Blender 3D computer graphics software. Blender is free and used for films, TV, and games, one example being the visual

Peter is a Senior Lecturer in Computing Education at the University of Roehampton. He taught Computing in secondary schools through the Teach First scheme, set up 3Dami in 2012 and currently conducts research in learning pathways of young digital makers.

CREATING DIGITAL ARTISTS
YOUNG PHOTOGRAPHERS

Creating future artists and photographers. Much more than just point and shoot

STORY BY Matthew Moore

For the past three years, I've had the pleasure of running a photography club for a small group of children. It's given me the opportunity to share one of my own passions with the pupils, as well as helping them to get creative more often. In my opinion, too much of primary school life is taken up by English and Maths! We need to give pupils the chance to let their creativity flow, and create something unique and memorable.

Each week, we meet up and spend an hour playing with a set of cameras in and around our school. I set them a challenge or a theme each week, and we head off to see what we can snap. It might be something as simple as giving them a title like 'Stone' or 'Shadows' and seeing what they come up with. Other weeks, we might go on short visits, or maybe look at the work of a professional photographer. The teaching comes from talking about photography concepts during each of these activities. We talk about lighting, composition, the rule of thirds, and why they think it's a good photograph.

Sharing is key

There's no point in the pupils taking all these wonderful pictures if they never get seen again! Our club has a few different ways of sharing their beautiful work. Firstly, at the start of the year, they're given an A3 art pad to be used as their portfolio. This is for them to keep and for them to use. Not a book that gets marked, but a pad that gets admired. It should be a reflection of their work, and space for them to show off their talent.

The second way was our own little blog space. The idea was for the pupils to keep it up to date and share their work. It looks great and is a nice way for them to share more of their work. The trick with this is to find a blogging platform with a nice storage allowance, otherwise you'll fill it up pretty quickly. This has also been a great way for us to share their work with the other parents at school. I can quickly send out the link in an email to parents for them to see the great work they do in the club.

The last way is a favourite of the pupils. Taking a wall in one of the main school corridors, we've created our very own Photography Club Gallery. They get to pick their favourite snaps (some of them would not be my choice!) and display them in frames, labelled with their names and camera used. All I did was buy the cheapest frames I could find at IKEA, and they love them! We have a three-month policy on changing them too, just to make sure the gallery stays fresh and new! They've even

been 'commissioned' to take some shots for the staff room walls by the deputy head teacher! This is such a positive experience for the kids, and is something they'll remember for a long time!

Simple equipment

Something important to remember when look at buying equipment for a club like this one… the cameras really don't matter! Honest!

We started off with some simple point-and-shoots. Bog standard, cheap, and cheerful. The trick was to get the kids thinking about how they could use them in interesting ways, and how they could make sure the subject in the photograph was captivating. After a few months, I managed to beg, steal, and borrow from the Head to get some Sony A5000s. These cameras are great. All the power of a DSLR in a camera, but they're small enough for little hands! These were a little pricey and fairly delicate – a scenario that will fill all primary school teachers with dread! So with the little funding I had left, I bought a few 'rough and tough' cameras. I chose the Ricoh WG-30. I can't recommend these cameras enough. Waterproof, shockproof, and they even have digital microscope capability. They really stopped me from worrying when the child with the clumsy

■ The DSLR pro photographers!

■ Cold and frosty morning! Use the macro setting on your camera for some interesting close-ups

■ Using the slow shutter speed on a DSLR to paint with light

■ A simple glass ball can create magical images

■ Broken equipment given new life in the hands of inquisitive minds

hands was running down the mud track in the middle of the Yorkshire Dales!

Allowing the children to investigate and explore with the cameras has been one of my drives for the club. One way I've done that is by collecting bits and pieces for us to use with the cameras. I've been to camera shops and bought old, broken, and scrap SLR lenses. These are great for looking at how a camera works, but also to hold up to the lens of your cheap camera and take some interesting snaps! I've also added a few small torches for painting with light. Have a Google about how you can achieve this as it's incredible! Using a slow shutter speed, you can paint using the light

of a torch. So cool! My personal favourite piece of equipment is a glass ball, though. Pick one up on eBay for around £15, and get ready to take some unbelievable pictures. They make even the most novice photographer look like a pro!

Three years on...

Having worked with the same small group of children for such a long time, I've seen them develop before me. The club has seen them change into future artists. It has instilled in them a sense of aesthetic. It gave them the opportunity to create something with the sole purpose of looking good. I'm not sure how often we get to

do that in a primary school anymore with the pressures of SATs. My biggest hope is that it's passed on a love and enthusiasm for photography in the pupils. That it's something they'll remember doing, and I can't wait to see some of their portfolios in future life! Fingers crossed! (HW)

Matthew is Computing Specialist in a primary school in Bradford. CAS Master Teacher and Hub Leader, as well as being a Raspberry Pi Certified Educator. I'm lucky enough to have a job where I get to just teach computing!

MORE MINECRAFT
WITH PHYTHON

It's time to have some fun with coordinates and teleportation

STORY BY Bob Irving

Welcome back to the wonderful world of coding Minecraft with Python! In our last article, we talked about how to get and install all the programs you need to code Minecraft with Python. Check back to *Hello World*, issue 6 and review the instructions there. Alternatively, you can find the instructions right here: **helloworld.cc/2rOV8Cn**

So we'll assume you've got Minecraft running as well as Python. You've got your Python shell window open, where Python sends you important messages. You've got your Python coding window open (by hitting File | New Window in the shell), and if you're on a Mac, PC, or Linux machine, you've got your Bukkit server running (if you're lucky to be on a Raspberry Pi, of course, you can skip that last step).

In the last article, we got our "hello world" program running, which produced some stunning output, saying "Hello World" in the chat in Minecraft. Sure, it's not going to become the next killer app, but we showed that we could connect our Python code to Minecraft and had it do something. And

that's a great first step! Now let's do some even more cool stuff.

We'll start by finding out where we are in the Minecraft world. As you probably know, all of that is tracked in the "coordinates". In the Java Edition, we can see them by hitting the F3 function key. We track our location in Minecraft by referencing the X, Y, and Z coordinates. Because Minecraft is played in three-dimensional space, we need all three. Basically, X tracks our position on the east-west axis, Z tracks it on the north-south axis, and Y tracks our up-down position. If we know all three, we'll know exactly where anything is in the world. And that's incredibly useful. In many games, the coordinate numbers measure pixel position on the screen. In Minecraft, the numbers measure blocks. So if our Y position is 5, that means it's five blocks above sea level. If Y is -10, then it's 10 blocks below sea level.

Let's do a little teleportation now, shall we? We're going to launch ourselves way up in the sky, then since we're in Creative mode, we'll fall back to earth unharmed.

And we'll get a beautiful view of the world beneath us as it rises up to meet us.

So here's how we do that. I've added comments for each line to explain what it does, prefacing each comment with a hashtag(#). That's our way of including information in the code that is meant only for humans. When the computer program Python sees a line starting with the #, it just skips that line. So that means that technically we don't need to type those lines for the program to run:

```
#import the Python Minecraft library
from mcpi.minecraft import Minecraft

#make a connection from my code to
the Minecraft game
mc = Minecraft.create()

#ask the game for the position
of the player and put it on the
variable called "position"
position=mc.player.getTilePos()

#now change the player's position.
```

> We're actually only going to change the Y position, which handles up and down

```
mc.player.setPos(position.x,
  position.y+150, position.z)
```

Once you've typed it in, run that code (Choose Run | Run Module) in your code window. Then jump quickly over to the game to see the view! And you've got to do that fast, because as soon as you run the code, it executes it, and in less than a second you've been teleported up about 150 blocks, and you'll start falling immediately!

For fun, teleport yourself even higher! Remember that the height of the Minecraft world is 256 blocks. So while you might think it could be fun to teleport yourself a zillion blocks in the sky, you'll crash your server! And that's no fun.

An important concept about how Python handles that variable "position". Most variables that are numbers are only one number, but remember that to know the position of anything in Minecraft, we need three numbers. So that variable actually holds three numbers, one for each of x, y, and z. In programming speak, we call that a "tuple". And we can refer to each of those numbers individually by using the variable name followed by the coordinate name: position.x, position.y, position.z. That's what allows us to only change one of them, which lets us go straight up 150 blocks and fall back to the spot where we started, because the x and the z positions didn't change.

```
#import the Python Minecraft library
from mcpi.minecraft import Minecraft
```

```
#make a connection from my code to
the Minecraft game
mc = Minecraft.create()
```

```
#ask the game for the position
of the player and put it on the
variable called "position"
position=mc.player.getTilePos()
```

```
#display the coordinates of the
player's position in the chat window
mc.postToChat("x="+str(position.x)
  + " y="+str(position.y) +"
z="+str(position.z))
```

Only four lines of code, but there's a lot going on there! And that fourth line is a really tricky one to type! It's super easy to make a mistake there, so if you get an error in the shell, go back and carefully go character by character through that fourth line to make sure that every mark of punctuation is totally correct. Remember that computers are not that bright! Humans can figure out what we mean even if it's not what we said. Computers only know what we said.

There are three key concepts in that fourth line as well. The first is kind of grammatical: those plus signs are how we put together an English phrase. It's called 'concatenation' and it just basically says 'join these together in this order'. It's not addition, though it's the same symbol. That's why you may have noticed the spaces within the quotation marks, which give us some space between words. The second concept we've introduced is those str's you see. 'str' is short for 'string'. In Python and almost all programming languages, a 'string' is what we would call text. We call this a 'datatype'. Python needs to know what kind of datatype we've got because when we display text (as we do in the chat window), we need to make sure that it's

text. The 'position.x, position.y, position.z' phrases are Python's way of grabbing those x, y, and z values we got in the line above it, where we got the player's position and put it on the variable 'position'. That kind of variable actually consists of three numbers (in programming speak, that's called a tuple). So putting str in front of position.x is Python's way of taking that number of my x position and changing it into text, so that I can display it in my chat window.

Whew! That's a lot of stuff right there. But it's all important programming, and you'll use these concepts many times. So it's essential that you understand them.

If this has whetted your appetite for doing even more cool stuff, please check out the resources above. And you can contact me by email: **bobirv@gmail.com**. I'm happy to help you in your Python and Minecraft journey! (HW)

> **Bob** teaches middle school computer science at the Porter-Gaud School in Charleston, SC. He's a Minecraft Global Mentor and a Raspberry Pi Certified Educator. More importantly, he provides 'hard fun' for his students.

THE TROUBLE WITH PSEUDOCODE...

John Woollard's personal reflection on the ongoing discussions regarding pseudocode

STORY BY John Woollard

"It is straightforward; it is simple; pseudocode is any representation, using text and numbers, of an algorithm. There are no other rules; it is whatever the programmer wants it to be."

In the classroom, a basic definition is:

"Pseudocode, text and numbers to represent an algorithm"

And to emphasise the point for learners:

"Pseudocode is your text and your numbers to represent your algorithm".

Pseudocode is important for programmers as:

■ It enables initial ideas to be recorded for the programmer's benefit

■ It enables those ideas to be communicated to someone else.

In the classroom, the second point dominates – we want to know that the pupils' ideas are sound before they start coding. Pseudocode enables them to express their ideas without the distraction of syntax and format.

When pupils are first introduced to programming, at whatever level, a popular example of an algorithm is 'making a cup of tea'. It's an activity associated with combining components in a specific order, over a period of time and ends when a condition is set:

```
place teabag in mug
boil some water
pour over teabag
wait until strong enough
remove the teabag
add some milk and stir
```

That's an example of pseudocode and represents someone's description of making a cup of tea. It would be evidence that the writer understood the sequence and components of making a cup of tea providing the reader knew how to make a cup of tea.

Pseudocode isn't necessarily executable. It's an abstraction with only the necessary detail required for communication.

The problem began with pseudocode when the Awarding Organisations (rightly) defined how their examiners would write pseudocode in the examination papers.

It's right that examiners should present algorithms in a standard and simple way. The practice ensures that candidates would not be disadvantaged because, although they could understand and apply a particular algorithm, they could not interpret a particular, idiosyncratic, representation of the algorithm. Another good reason is that it enables the proof reader of papers to more easily detect errors in the logic of the examples. (In Scotland, examination papers use a particular format for the algorithms that can be checked using the Haggis program.)

The problem then manifested itself in the classroom when some teachers thought that pupils had to write their algorithms in the same format as the examination papers. That is NOT true. There's no specified format for candidates to use when answering questions. Pupils don't have to learn to write in a particular format. Their answers simply have to be clearly stated.

For example, "Questions in the written examination that involve code will use this pseudocode for clarity and consistency. However, students may answer questions using any valid method."

The problem would have been avoided if the term pseudocode had not been used to describe how algorithms would be presented in the examination papers. The term 'reference language' would have been good.

Pupils would then have learnt what pseudocode was and could have written their own algorithms using their own pseudocode.

Examination writers, authors of textbooks, and teachers creating their own resources would have used the reference language to write out algorithms.

But for some, the problem continues. The Awarding Organisations chose to use different formats to represent algorithms in their papers. Schools changing courses or pupils changing schools might have to become familiar with a different way of reading algorithms.

For example, an examination paper might represent the algorithm for calculating the area outside of a disc. One paper would be very different from another:

```
#area outside of disc
PRINT "Enter size"
RECEIVE size FROM (REAL) KEYBOARD
SET square TO size * size
SET radius TO size * 0.5
SET disc TO PI * radius * radius
SET area TO square - disc
```

Compared with:
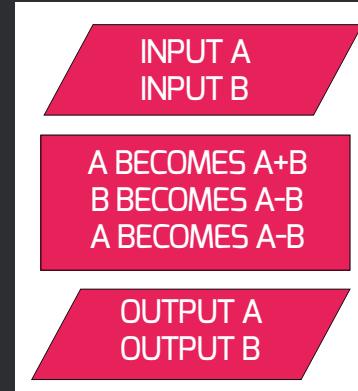
```
//area outside of disc
size=input(Enter size)
square=size*size
radius=size/2
disc=3.14*r^2
area=square-disc
```

```
And finally, on a good note, using
pseudocode to support annotation.
```

Pseudocode is text and numbers to represent an algorithm; they can be handwritten or typed. By encouraging the pupils to type their pseudocode directly into the development environment, they'll automatically and naturally be creating the annotation for their program.

For example, the initial outline might be:

FLOWCHARTS

Flowcharts are another way to represent algorithms. There's no single definitive flowchart specification.

Flowcharts are visual representations of algorithms. Many symbols are common and represent the programming constructs of sequence, selection, and iteration with input, output, and process, and representations of data storage. Most word processors and presentation software contain the standard shapes.

```
# declare the variables
# open the file
# keep repeating until last person
# input the value
# write to file
# process the data
# output the result
# close the file
```

This outline can be discussed with the teacher or programming partner, and extra detail added as ideas become clearer. Code can be inserted and developed.

Pseudocode is a powerful tool for computer programmers. Its individual, free-form nature to represent the individual programmer's thoughts must not be contaminated by requirements of syntax or format.

The use of a reference language by Awarding Organisations to direct examination writers and guide those creating textbooks and teaching resources is welcomed.

But the two must not be confused – they are NOT the same. (HW)

ANOTHER EXAMPLE OF PSEUDOCODE...

```
procedure maketea
objects:mug,teabag,water,milk;
actions: pour, put, boil, stir,
drink;
conditions: strong, cool;
begin
boil water;
put (teabag, mug);
pour (water, mug);
repeat [ ] until strong = TRUE
put (teabag, NOT mug);
pour (milk, mug);
stir (mug);
repeat [ ] until cool = TRUE
end procedure
```

This is obviously not programming code - there isn't an interpreter that could handle such a structure, but it also clearly has a structure and syntax, and has been written by someone with a little programming knowledge. A person's pseudocode will change over time as he/she gains knowledge of particular syntax or structures. Again, this text provides evidence that the writer understood the sequence and components of making a cup of tea providing the reader knew how to make a cup of tea.

Alternatively, this text can be used as a teaching resource - asking pupils to read and explain what it means would help develop their programming literacy. Asking questions like, "what is the instruction to add sugar?" starts to encourage a consideration of syntax.

THE KEY TO HELPING NOVICE PROGRAMMERS: LANGUAGE ACQUISITION INSTRUCTION

Introductory programming instruction can reach more, if not all, students if supplemented with pedagogies that address the acquisition of programming languages as languages per se

STORY BY Scott R Portnoff

Instructors of any introductory computer programming course routinely observe two groups of students: those who learn and progress (the traditional demographic overwhelmingly composed of white/Asian males) and a sizeable group who, though at grade level, struggle throughout and learn practically nothing. Known as the Novice Programmer Failure Problem (NPFP), over four decades of efforts to improve outcomes for the strugglers have proven unsuccessful. At the secondary level, the phenomenon is even more lopsided and intractable, affecting the majority of grade-level students. In the United States, secondary CS instructors (90% of whom it is estimated have no formal CS

education), like all public school teachers, are tasked with delivering effective instruction to all students, thus putting them in an existential bind. About a dozen years ago, a small group of educators decided that programming education – i.e. the Advanced Placement Computer Science A course (APCS-A) – was simply not feasible.

The survey courses

As response, two secondary survey courses, Exploring Computer Science and Advanced Placement Computer Science Principles, were developed with National Science Foundation (NSF) funding. Proponents viewed them not as a solution to the NPFP, but as a way to skirt it

altogether. Attacking the 'programming-centric' focus of the APCS-A course as exclusionary, they concocted a novel narrative in which programming was simply one of several co-equal strands in the secondary CS curriculum. Despite the rhetoric, both courses are considered pre-APCS-A courses and have proliferated in low-performing urban public high schools.

Proponents argued that the survey courses would 'broaden participation' beyond the traditional APCS-A demographic to females and traditionally under-represented minorities. NSF's goal was more pragmatic: to create a high school entry point into a CS pipeline intended to alleviate a massive number of projected computing vacancies.

At the same time, however, a College Board study found that students taking the APCS-A exam had a six- to eight-fold higher probability of choosing a CS college major (Morgan & Klaric, 2007). A later study found that 20% of students who score 2 or higher on the exam choose a CS major, and that fully 27% of students earning a 5 go on to major in CS (Mattern, Shaw, & Ewing, 2011). Thus the APCS-A course was identified as being the entry point into NSF's CS pipeline, an inconvenient fact ignored by the new narrative.

Redefining 'participation'

The survey courses intentionally avoided attempts to deliver rigorous programming

instruction – doing nothing to prepare students for subsequent CS coursework that would inevitably involve programming – opting instead for superficial hobby-like treatments of programming and other topics. As these courses proliferated, the effect over the past decade has been to greatly lower academic expectations for secondary CS education. Instead of helping students master content that vertically aligns with subsequent academic coursework, mere exposure to simplified topics is expected to generate interest that will magically enable students to later circumvent the NPFP and acquire programming competence.

Unlike APCS-A, there exists no research demonstrating that the survey courses spur students to take – or prepare them to pass – the APCS-A exam or future CS college coursework. Moreover, the survey course narrative that demoted programming from its central place in the introductory curriculum is a fiction. The universal post-secondary consensus is that programming is the core skill fundamental to the entire discipline – crucial for understanding and plumbing both basic and advanced CS topics including algorithms – on anything beyond a trivial level.

Instead of authentically 'broadening participation', therefore, the survey courses have simply swapped one unequal two-tier track system for another, ironically, but predictably, recapitulating the very inequities they were intended to remedy. The facade of more high school students taking classes labelled 'Computer Science' may have PR value, but it's only the latest in a long history of ineffective 'innovations' that continue to obstruct, confuse and delay real reform.

That being said, APCS-A is not without its own problems. The course may be the CS pipeline entry point, but it's simply too advanced for most grade-level students and demands a prerequisite. Why CS educators then put their efforts into developing a survey course is a mystery, because it's unremarkably obvious that an introductory course should be a programming course that will effectively identify and address head-on the problems novice programmers encounter.

The critical pedagogic role of language

In this regard, three recent fMRI studies have confirmed that comprehension of computer programs occurs in the same regions of the brain that process natural languages – not math, not logic (Siegmund et al, 2014) (Floyd et al, 2017) (Siegmund et al, 2017). This cognitive-physiological evidence indicates that programming languages, despite being artificial languages, are alive in the brains of programmers in much the same way as any natural language that those programmers speak. This is a profound paradigm shift for thinking about how students learn – and are taught – programming languages, and supports a compelling argument for investigating pedagogies that address language acquisition factors. I've previously described supplemental instructional strategies that focus on the teaching of programming languages as languages per se, and that, importantly, reach grade-level students who formerly would have learned little (Portnoff, 2016 / 2018). I therefore contend that the root cause of the NPFP is a pedagogic gap – we want students to use logic to solve problems, but we neglect to provide instruction that will help them acquire the very language which mediates that logic.

Human language is an innate, highly specific cognitive ability quite distinct from general intelligence. A language can't be taught explicitly; for example, through grammar instruction. Rather teachers need to provide the conditions, situations, and experiences that facilitate its gradual acquisition implicitly through: (a) repetitive exposure to language data – vocabulary, syntactic patterns, and paradigms – in meaningful contexts; and (b) intentional expressive use of the language with implicit feedback. For most students, the start of meaningful automaticity and basic programming fluency – like the timeline for foreign languages – emerges after two to three years. Expectations for acquisition of programming skills therefore need to be adjusted and the timeline extended from a single course to a multi-year course sequence.

Next time: Supplemental instructional strategies that can help students acquire programming languages implicitly. (HW)

Scott teaches Computer Science at Downtown Magnets High School in the Los Angeles Unified School District. In 2013, he wrote a contextualised introductory programming course that uses models and simulations for exploring real-world cross-curricular topics spanning the fields of art, geography, astronomy, political science, and biology, into which he is currently integrating language acquisition strategies. A unit from this course, Dynamic Word Clouds, resides in the Engage-CS-Edu repository and earned an Engagement Excellence designation.

'HELLO WORLD' CONSIDERED HARMFUL

No, not this magazine, but the idea of teaching programming by introducing user interaction before writing functions. It should be the other way around

STORY BY Richard Pawson

Your very first program probably looked something like this (allowing for differences between programming languages):

```
A = INPUT('Enter a number')
B = A * A
PRINT('The square of the number is: ', B)
```

Maybe you then added a loop, a condition, or a more complex calculation. But from the very beginning, you were taught to mingle user interaction with instructions that transformed data, and you were taught the interaction bit first: `PRINT('Hello World!')` perhaps. At some later point, you were introduced to subroutines/procedures and, depending on when you were taught to program, functions (neither Fortran nor BASIC had functions when I learned them). Hopefully, you were encouraged to extract the calculations into functions, so that they could be re-used, or perhaps to make testing easier, though you seldom did reuse them, or test them in isolation, so it seemed like a lot of work for nothing.

Well, the good news – depending on your perspective – is that the way we teach programming hasn't changed much since the early 1980s. But I suggest that this is an anachronism, and it needs to change: today, we should teach children to write functions first, and only then write 'programs' to invoke these functions, wrapped up with some more friendly user interaction. When I proposed this on the CAS forum recently, it invoked a lot of response, quite a bit of it negative. But there were some teachers who agreed; others who said that while they are not very sure about it, deep down something about the proposal struck them as right; still others that they would like to but don't want to fight city hall – the exam boards and book publishers who show no awareness of the issue. (Of particular annoyance to this author is AQA's total disregard for this issue in its approximately 1000 lines-of-code 'Skeleton Programs' that pupils must study as English students study *Romeo and Juliet*.)

Separation of concerns

The 'separation of concerns' is one of the most important principles of software design, recognised by most professional developers. It applies at many levels, many

> **"** TEACHING PUPILS TO WRITE FUNCTIONS FIRST, THEN BUILD INTERACTION AROUND THEM, WILL EASE THE TRANSITION TO FP

scales, and in many different ways, but is perhaps best explained in one simple, and common, form: a unit of code should be concerned with the user interface, or with application logic, but never with both. At A-level, where computer science students typically develop a substantial application, I teach mine that the separation of concerns delivers tangible benefits within the life of

the project. It allows them, for example, to start with a simple console UI then switch to a GUI without rewriting the whole application; or to switch from file-based persistence to a database. I find that getting students out of the habit of mixing the concerns – taught them from the outset – is hard.

Worse, I find that many students are coming into A-level with a high grade in GCSE Computer Science, yet with a very poor understanding of the idea of a function: "Sir, why does it give me an error: 'Not all code paths return a value'?" "Because your function is not returning a value when it should." "But it does return a value, here … Console. WriteLine(result)" "Writing to the console is not returning a value."

Functional programming

A growing proportion of both the academic and professional programming communities now believe that 'functional programming' (FP) will become the next dominant paradigm in programming, just as object-oriented did around 20 years ago, and structured programming before that. Two years ago, AQA introduced FP into its A-Level syllabus, presenting a significant challenge for many teachers. Suffice to say that a clear understanding of functions is just the start point.

Learning Haskell (arguably the purest of the FP languages), and how best to introduce it at A-level, involves embracing many new ideas, some of them very complex. But at a simple level, I've been struck by the fact that when teaching Haskell, you write, and use, functions long before you write any interaction. Here's the equivalent to lesson 1, in Haskell:

```
square a = a * a

square 5
=> 25
square 7
=> 49
```

Not only do you write functions first, but, from the outset, the idea of the difference between a parameter definition (a, above) and the arguments that you pass into those parameters (5,7) is made clear –

```
Program Hello;
Begin
Writeln('Hello, World!');
End._
```

another thing that many students struggle with when they try to 'extract' functions from their just-make-it-work program.

Read Evaluate Print Loop

The difference in Haskell is due not just to the language, but to another development in programming that has gained a lot of ground recently: REPL, which stands for Read Evaluate Print Loop. The environment (tooling) does the input and printing, transparently: you don't have to write INPUT or PRINT to use the function(s) you've defined. (If this reminds you of LOGO, that's not a coincidence.)

Only later in Haskell do you learn how to build functions into 'actions' which handle Input/Output programmatically, and you're forced to keep the two concepts separate, because actions have 'side effects' (some people call them 'dirty functions') and if you try doing input/output within a normal function you'll be forced to acknowledge that you've created a dirty function.

Despite having built more than half my career on the object-oriented programming paradigm, I am coming round not only to

the view that FP will become dominant in the commercial world, but that within a few years we'll be teaching FP as the introduction to programming in schools. (I believe it's too early to start now – the tooling for Haskell just isn't friendly enough, though there are researchers working on more child-friendly tooling for Haskell and other FP languages.)

Teaching pupils to write functions first, then build interaction around them, not vice versa, will make the transition to FP easier, for both pupils and teachers. As I've hopefully demonstrated, though, even if you don't understand, or buy into, the FP paradigm, this inversion of our current practice would have clear and tangible benefits even in the context of conventional procedural programming. (HW)

Richard has worked in the computer industry for 40 years, and claims he was the very first person in Europe to write and run a program in Microsoft Basic. Two years ago, he decided to become a teacher, and now teaches A-level Computer Science at Stowe School.

LEARNING LANGUAGES IN SCRATCH

Introducing the Google Translate and text to speech extensions for Scratch 3 to develop a drill and practice program for learning (almost) any foreign language

STORY BY Miles Berry

Lots of educational software follows essentially the same algorithm, which might not be that different from what teachers have done in their lessons:

```
Repeat until you've asked
enough questions:
    Ask a question
    Get a response
    If the response is right:
        Say 'well done'
```
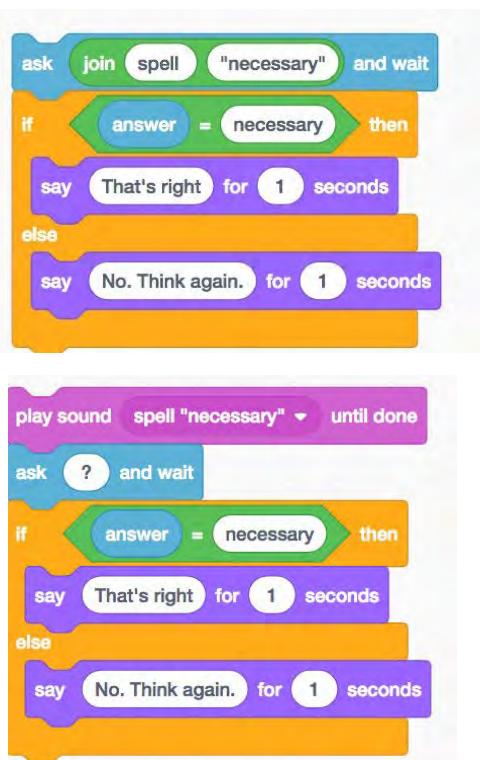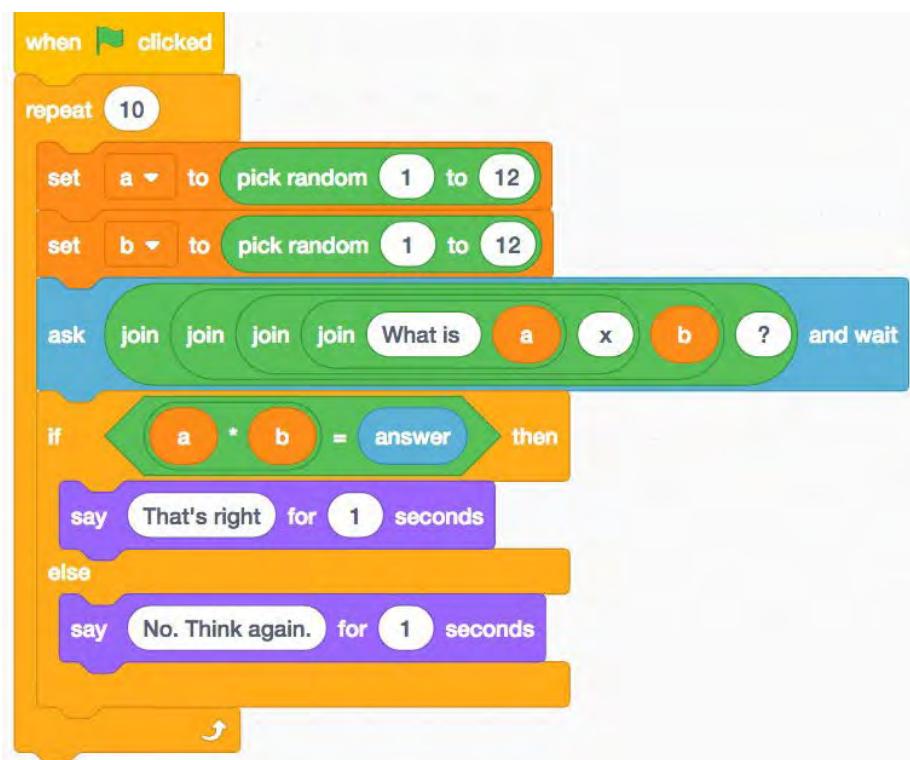
```
Else:
    Say 'no, think again'.
```

It's a nice programming challenge to try to write something similar in Scratch, perhaps with pre-made questions, or using the random number block to generate questions automatically (see below left).

One of the lovely things about this simple program is that it uses sequence, selection, repetition, variables, input, and output – all the constructs needed for the programming requirements of the English computing curriculum, making it a great example.

It's also highly remixable – we can take this outline of code and make it a 20-question test, or keep track of the score, or work on the user interface so it's just a bit more funky than text on screen, or any number of other possibilities. For example, as well as

testing tables, could we use code like this to test spellings? Well, maybe. Asking the question is easy enough (see opposite page).

But alas this is now just a little too easy as a spelling test.

We could record some audio and play that, but it's not a very flexible bit of code (see opposite page).

But now in Scratch 3, we've access to text to speech blocks as one of the standard extensions (libraries). Click on the button at the bottom left of the window and then select the text to speech extension (see top left).

So what about applying this idea to learning other languages? Well, clearly we can just check answers to questions (see top right).

We can go further than this, though, using the Google Translate extension/library to look up the word in our target language, and ▶

TAKING IT FURTHER
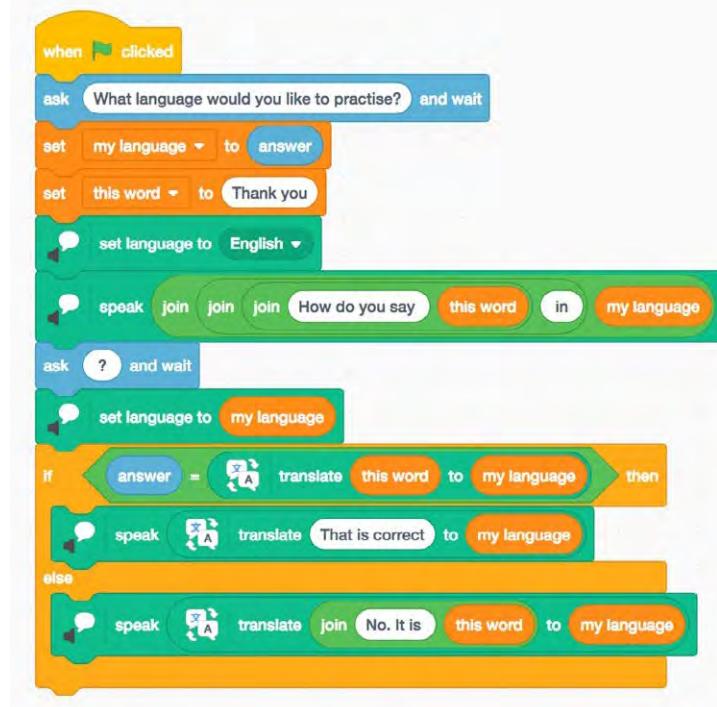
There are plenty of ways to improve the user interface here. What might you do?
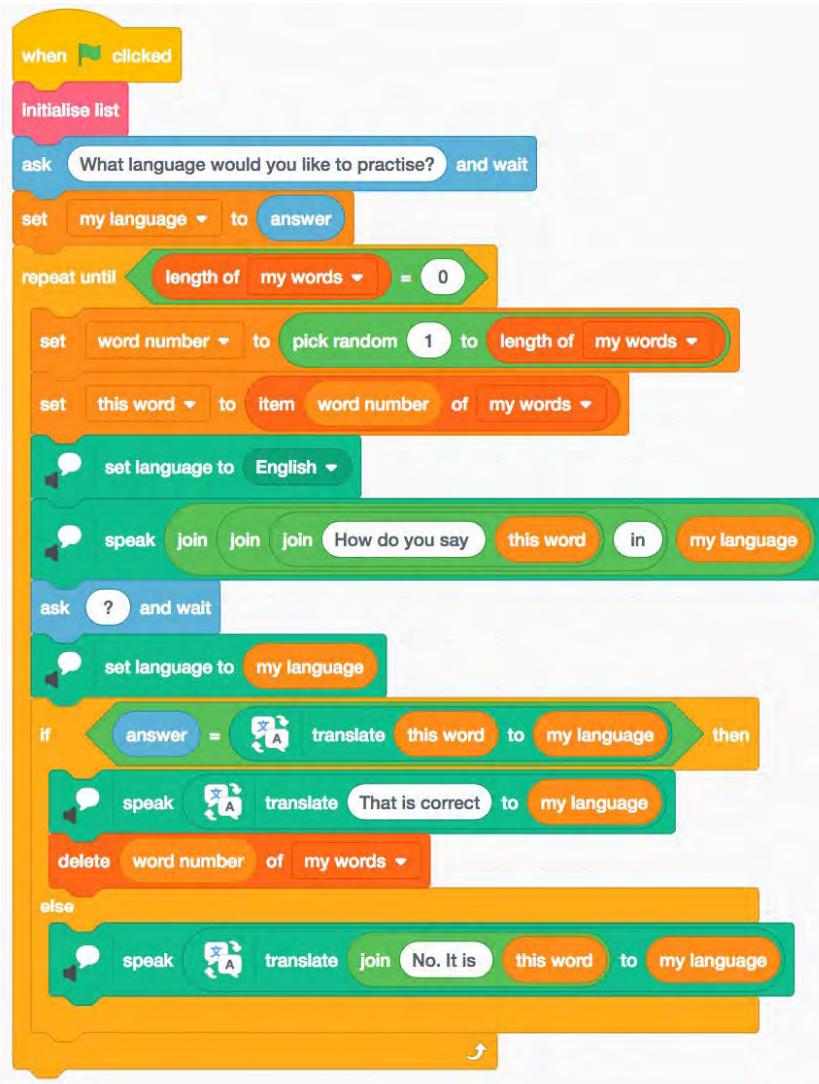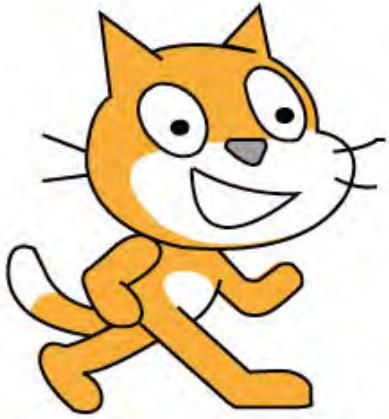
✔ Can you keep track of the player's score, or impose a time limit?

✔ How does Google translate work? Is there any way to allow for more 'fuzzy' translations? Should 'danke' and 'vielen dank' both be awarded the points for translating 'thank you' into German?

✔ The Scratch team have been working on voice to text - how could you incorporate that in your code?

✔ Experiment with Scratch's language packs, and try reading, editing or even writing the program in another language!

```
define  initialise list

delete all of  my words ▼

add  red  to  my words ▼

add  green  to  my words ▼

add  blue  to  my words ▼

add  yellow  to  my words ▼
```

```
when ⚑ clicked

initialise list

ask  What language would you like to practise?  and wait

set  my language ▼  to  answer

repeat until  < length of  my words ▼  =  0 >

  set  word number ▼  to  pick random  1  to  length of  my words ▼

  set  this word ▼  to  item  word number  of  my words ▼

  🔊 set language to  English ▼

  🔊 speak  join  join  join  How do you say  this word  in  my language

  ask  ?  and wait

  🔊 set language to  my language

  if  < answer  =  🗛 translate  this word  to  my language >  then

    🔊 speak  🗛 translate  That is correct  to  my language

    delete  word number  of  my words ▼

  else

    🔊 speak  🗛 translate  join  No. It is  this word  to  my language
```

then the program can be used to test any language, or at least any of those that Google Translate knows about (see bottom of previous page).

It's worth pointing out a couple of other refinements in the code here. Notice that we're giving the feedback in the target language rather than in English, and if the user gets the translation wrong then we helpfully tell them the correct translation, in the chosen target language. We also change between English and the target language for the text to speech blocks, so that the pronunciation for the question and for the feedback is correct. It's quite fun to experiment with the different language settings here, for example using French to pronounce English…

The last stage is to do something about the range of vocabulary we're testing. The version above just tests "Thank you", which is certainly important vocab, but typically you need a few more words to get by. Let's introduce the idea of a list as a way to keep track of all the words we need to practise, initialising this when we start the program with a custom block (or you could use a broadcast message instead) – see above.

We can then simply pick our target word from the list. Note that our code here removes words from our vocab list when we get them right, but leaves them there for another go if we get them wrong (see opposite). (HW)

Miles is principal lecturer in computing education at the University of Roehampton and a former teacher. He serves on the boards of CAS and the CSTA and is a member of the Raspberry Pi Foundation.

AGE RANGE

11-12 years

LESSON TYPE

Programming

REQUIREMENTS

• Scratch

CLEVER CAT MACHINE LEARNING IN SCRATCH

Teach your students that Artificial Intelligence and Machine Learning are not obscure and inaccessible, but simply an extension of what they can already do

STORY BY David Horton
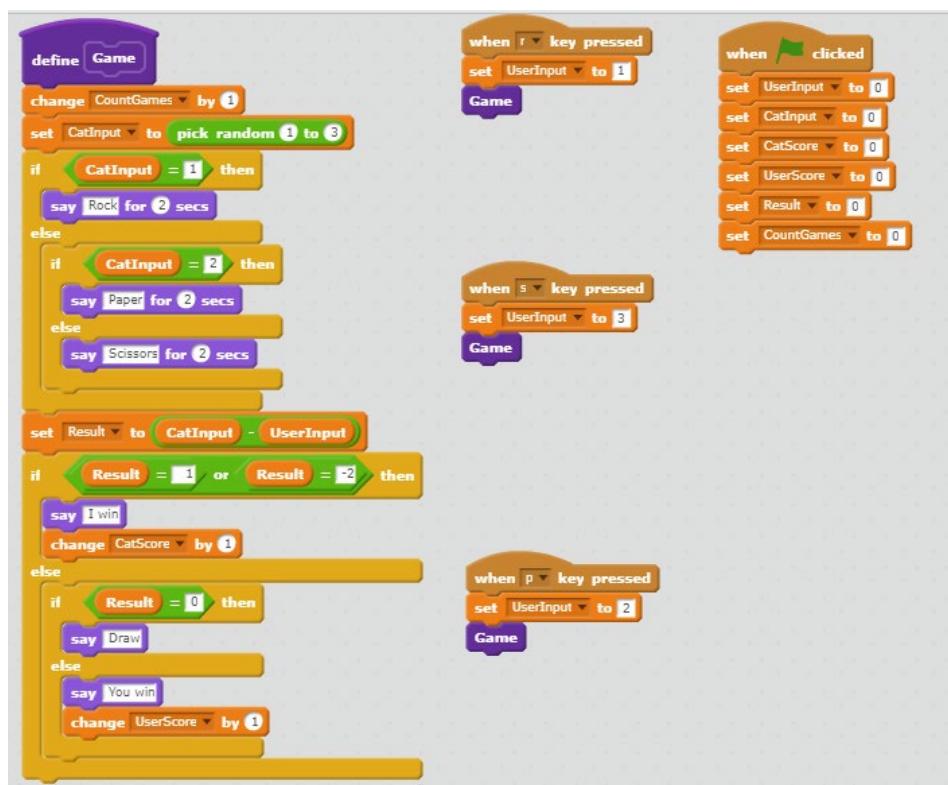
I was recently at an event with the venerable Miles Berry, editor of this lovely publication and all-round gentleman and scholar, who was having to be the servant of two masters as he battled with a dual brief of speaking on Computational Thinking and AI in the same slot. As one might imagine, he did a great job and, as I sat at the back, I started to mull over how I could introduce the children to the idea of actually programming something which exhibits AI/ML behaviour, rather than just talking about it or demonstrating pre-existing systems.

So far this term, I've been working with a small group of 11- and 12-year-old coders using the excellent *Cracking Codes with Python* online book, and felt I had taken them about as far as I could in that direction. I figured then that they would be good guinea pigs for the exercise. I chose to attempt it using Scratch for a few reasons:
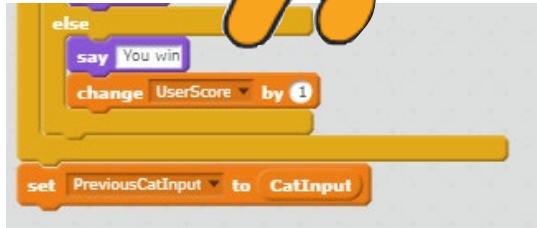
1. Scratch is pretty ubiquitous, and I had half a mind of sharing the process
2. I wanted to use a platform where the code wouldn't 'get in the way' at all
3. I wanted to emphasise the simplicity of the concept, and it doesn't get much simpler than our favourite orange cat

We started out by digging into what Artificial Intelligence really is in its most fundamental form – that is, a system with enough knowledge of its environment to be able to make autonomous decisions. This leads nicely into the more helpful idea of Machine Learning – a concept more within the reach of the average 12-year-old.
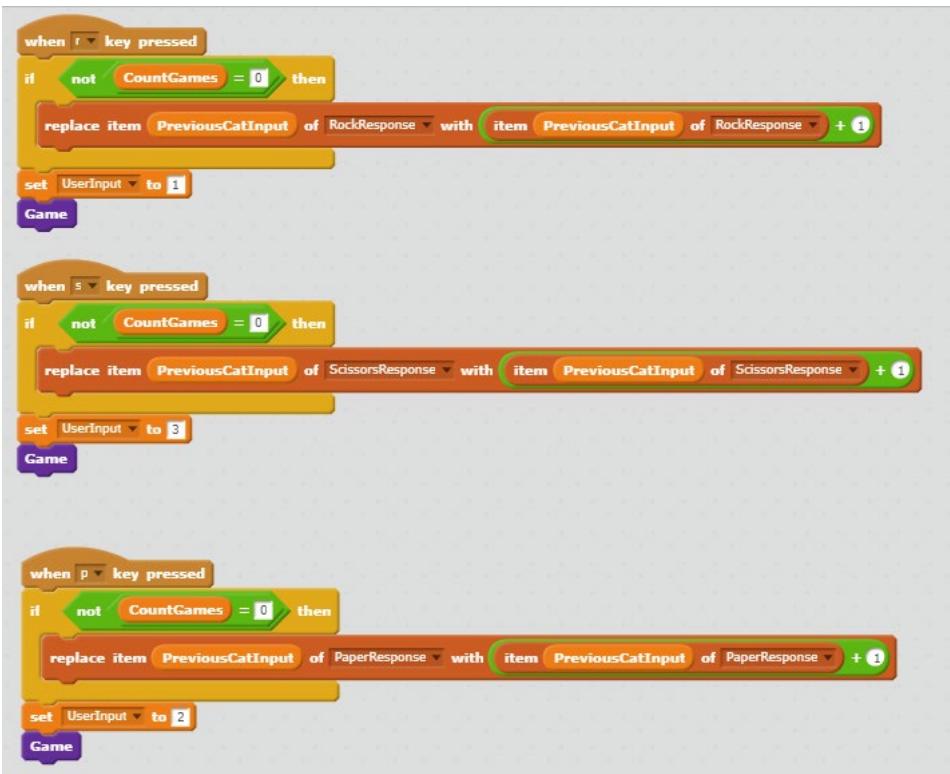
We needed a simple game we could play against the computer, and settled on Rock, Paper, Scissors. Easy to code in its basic form, with very clear rules, but various options around strategy, it fitted the bill perfectly. I created a form of the game using the simplest commands I could. The user simply hits R, P or S, and the computer responds with a random choice. It then makes the judgement as to whether the round is a draw, or a human or feline win.

As you can see, it's simple stuff. We simply assign the values 1, 2

▶ and 3 to Rock, Paper and Scissors respectively, and then call a function to see who's won. I appreciate that many people wouldn't consider defining a block as the most basic of functions, but I strongly recommend to get pupils into the concept as early as possible. It is, after all, how most programming actually happens in the real world, and once the idea is embedded it makes life much simpler.

The main block is pretty self-explanatory: we increment a counter to keep track of rounds played, and then get the cat to pick a play, then tell us what it is. There's a number of ways of doing the next bit, but I chose the reasonably efficient method of subtracting one from the other. A zero means a draw with a 1 or -2 a win for the cat. There's a teaching point in that somewhere, I'm sure, but I'll leave you to pick the bones out of that one. The result is announced and we move on.

All we need to do then is initialise our variables on the green flag and we have a working game.

So far so good. The next thing is to get our program to start learning.

There was a range of possible options as to which aspect to focus on, but I figured the single most influential factor on a player's decision-making was the opposing player's previous choice. This took some thinking about, but it required recording the cat's choice against the player's next choice. Since there are three possible options for each of these, a two-dimensional array would have been the best way forward, and the simplest option would have been nine variables. As a compromise, along with some additional teaching, I went for three list variables, plus a variable to record the previous guess from the cat. Skipping the first play (when there's no previous guess), this code increments the list using the previous guess as a locator. This might take some explanation, but is a really good teaching point which will help the pupils understand how to address arrays in the future.

You can show the lists on screen to see what is happening. Again, it requires a little thought as it's not unlike the brilliant *Two Ronnies* Mastermind sketch where the specialised subject is answering the question before last. There are slightly less involved ways of achieving this recording, but I think the combination of learning the list variable with the broader options this method presents for future development of the program justify the complexity. One interesting challenge that the list variable presented was resetting at the start of the game. To be able to address the position in the list, it first has to exist, so we had to delete all the previous elements and then rebuild the list every time we restart the game. Here's the blocks (see right) to do it (note that unlike an array, the first position in the list is addressed as 1 not 0.)
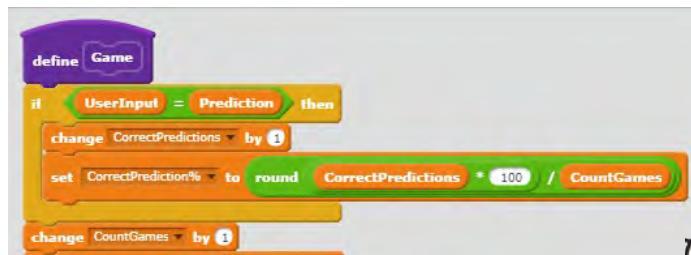
The final piece of the jigsaw, at least for now, is to turn our new-found knowledge into something that will demonstrate learning; after all, learning is the acquisition and application of knowledge, so why should we think of ML any differently? I chose to do this by getting the cat

to start predicting the player's next move, since this could be done quietly behind the scenes and hopefully without excessively influencing the human user to try to out-think us. This was a two-stage process. Firstly, we had to see if we had enough data to make a prediction, and then determine the most likely next play, so these blocks went in at the bottom of our Game block (see top of page).

I'd be lying to you if I didn't say I was having to resist the Siren call of our favourite serpentine friend's text-based code here, but the point was to keep it simple and obvious, so I stuck with it. The final very easy stage was to test our prediction and keep score of our success. I decided to turn it into a percentage, since it's easy for the children to understand that 33% is what we'd expect from random prediction, so any significant improvement on that (say about 50%) demonstrates that the machine is being more intelligent than average. I'm sure you could work this out, but for the sake of completeness see above.

If you're making this up for yourself, then don't forget to **initialise** all these variables to zero on the green flag.

I suspect that those of you who have journeyed this far with me today (well done!) will already be leaping off ahead with loads of ideas about how this could be developed. It's one of the things I love most about working with teachers: once that little spark of an idea lands, it quickly turns into a conflagration of teacher inspiration. I expect you're already at "we could extend this to take account of the user's previous guess" or "how does the previous results (win/lose/draw) affect the play". If, like me, you're very competitive, you might want to go further and start applying these predictions to beat the human, or even think about testing the human with certain combinations in order to reduce gaps in your data. If
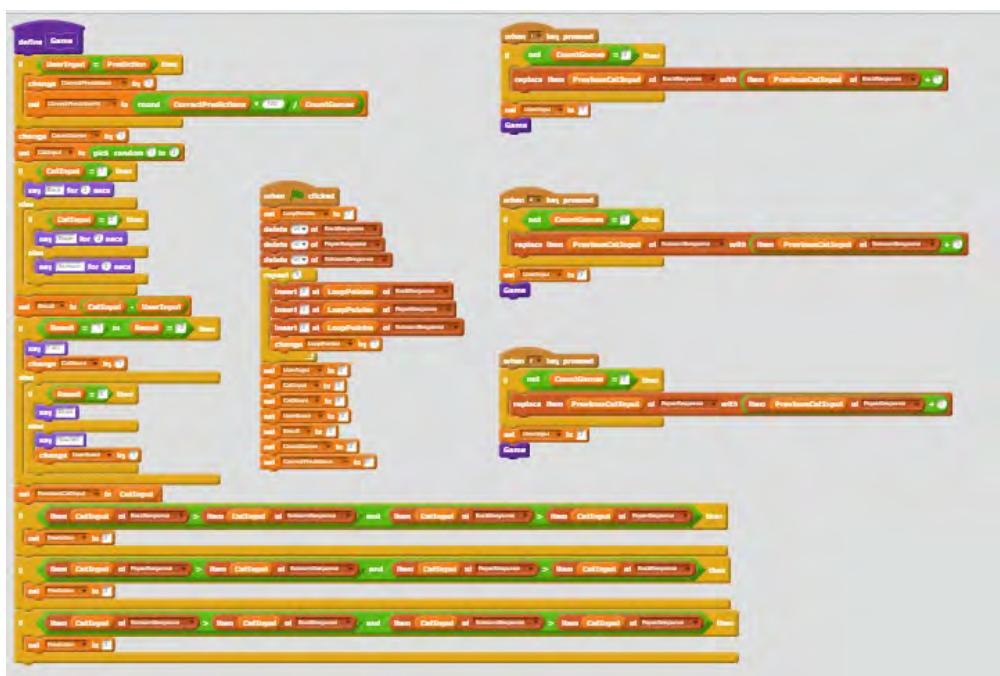
so, then you're thinking like an AI/ML pro.

These are certainly conversations you can have with the children, even if you choose not to pursue the code itself. Some may go home and play with it, or translate it on to their preferred platform, but the important message to get across is that AI and ML is not some mystery like quantum physics which will require years of effort to understand, – it's just getting a computer to acquire and then apply information. Here's the zoomed-out view of all of the code (see left). About half is gameplay and

the rest is the ML bits. It's really very manageable.

As I mentioned above, I'm sure you already have ideas to develop this, so do share any you're pleased with. In conclusion, whether or not you're actually going to use this idea, I hope the clear message is that AI and ML aren't some obscure and inaccessible science of their own, but simply an extension of what we can already do.

Getting your children used to this idea not only enfranchises them into the rapidly developing world around them, it may even place them in the driving seat. (HW)

AGE RANGE

15-18

LESSON TYPE

Programming

REQUIREMENTS

• Unity

GETTING STARTED WITH UNITY

In this project, you'll make your first game with Unity, a professional game development platform

· **STORY BY** Janina Ander ·
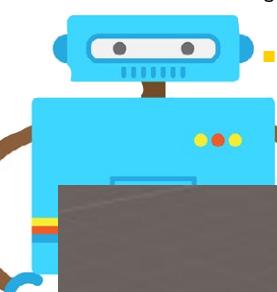
Introduction

You're going to build a simple video game with a robot, a ball, and a maze, and you'll learn the tools you can use to make the game bigger and more awesome! What you'll learn:

- How to create 3D objects
- How to change the colour, shape, and position of objects
- How to adjust the camera angle to follow a player in the game
- How to add a script to an object, and write code to control movement and behaviour

Set the stage

Before you can start to make your game, you need to do a little setup:

- Start Unity and choose *New* to create a new project.
  - When asked for a name for the project, call it "Beginner
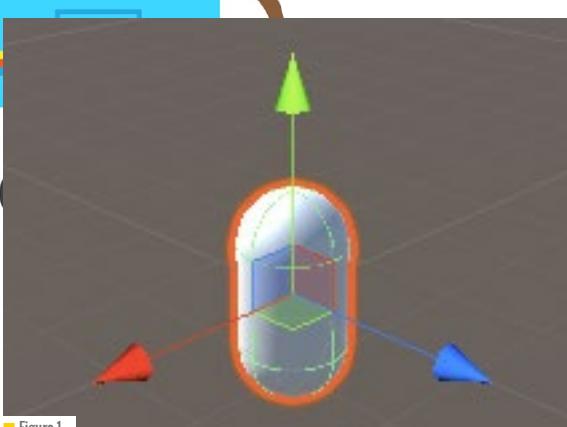
Unity Sushi" so it will be easy to find later!
- Click *Create Project* and wait for your project to appear on screen!
- In the top right-hand corner of the Unity window, you'll see a menu called *Layout*. Click on it and pick *Default*.
- Look at the panel named *Assets*. You should see a folder there called `Scenes`.

The next thing you need is a scene. It turns out you already have one, because the Unity project started with one, so you just need to save it:

- Go to the top menu and choose *File > Save Scene as...*. You'll need to give the scene a name and choose a location to save it in. Call it `MazeRoboBegins` and save it in the `Scenes` folder that's inside the `Assets` folder.

Make a robot

Time to create your first object:

- Make a *Capsule* object (*GameObject*

> 3D Object > Capsule): this will be the body of your robot! [Figure 1]
- Select the Capsule by clicking on it. On the right, you should see loads of options and menus. This is called the *Inspector*, and it's where you set up most of the objects in your game.

You can rename an object by typing a new name in at the top of the Inspector:

- Change the name of the Capsule to `MazeRobo` now.
- Next, to be sure that MazeRobo is right in the middle of the game world, look in the *Transform* section of the Inspector, click on the cog icon, and choose *Reset* [Figure 2].
- You need a couple more objects to make your robot, so create a *Cube* (*GameObject > 3D Object > Cube*) and a *Sphere* (*GameObjects > 3D Object > Sphere*).
- Change the name of the Cube to `Shades`, and the name of the Sphere to `Nose`.
- Look at the left of the screen. You should see a list of the objects in

■ Figure 1

RESOURCES

This lesson plan is taken from the Raspberry Pi project collection - work through online at `helloworld.cc/2Vsivz4` for more screenshots and extra hints!

Unity is available for free from `helloworld.cc/2R8p8rZ`

your game, including `MazeRobo`, `Shades`, and `Nose`. Click on `Shades` and drag it onto `MazeRobo`. Then drag `Nose` onto `MazeRobo` in the same way.

- Now select the `Shades` object and look at the Inspector's *Transform* section. You'll see a set of three *coordinates* (X, Y, Z) that control the object's *Position*.
- Try changing each of the coordinates' value to see which direction they control. Try putting a – in front of some of the numbers, too! Finally, set them to these values:

```
X = 0
Y = 0.64
Z = 0.42
```

- Do the same for `Nose`, setting them like this:

```
X = 0
Y = 0.5
Z = 0.5
```

This doesn't quite look like anything yet, does it? To make MazeRobo look like a robot, you'll adjust what `Shades` and `Nose` look like. You can control the shape of objects with the *Scale* controls:

- Staying in the *Inspector*, look at the scale controls for `Shades`. Set its scale to these values:

```
X = 0.64
Y = 0.16
Z = 0.16
```

- Now set the `Nose` scale to:

```
X = 0.16
Y = 0.16
Z = 0.16
```

Now it's starting to look like a robot!

Add colour

- Create a new folder by clicking *Assets > Create > Folder*, and call it `Materials`.
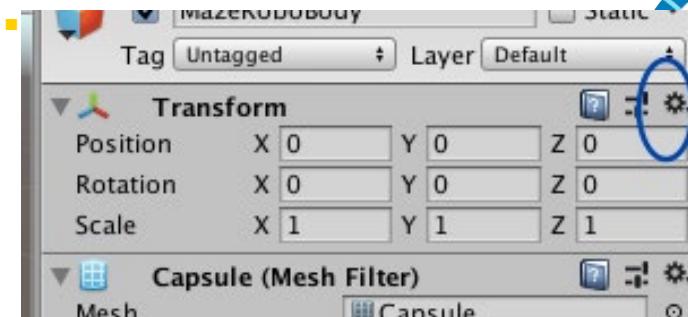
Now make two *materials (Assets > Create > Material)* called `EyeBlack` and `NoseRed`.

- The two new materials should be in the `Materials` folder you just made, inside the *Project* pane at the bottom of the screen. If they're not there, drag them onto the the `Materials` folder to place them inside it.
- You can set the colour of a material by changing its albedo value in the Inspector. Click on the rectangle next to the dropper icon, and a colour picker should open.
- Make `EyeBlack`'s *albedo* value black, and `NoseRed`'s albedo value red.
- Select the `Shades` object, look at the *Mesh Renderer* section of the Inspector, and expand the *Materials* subsection. Click on the small circle to the right of *Element 0* and select `EyeBlack`. Now MazeRobo has black shades! [Figure 3]
- Do the same for the `Nose` object as you did for the `Shades` object, only now select the `NoseRed` material. Now you've given MazeRobo a red nose!

Giving your robot rules

MazeRobo needs a *Rigidbody* component so you can move it about and let it interact with the world:

- With MazeRobo selected, click on *Component > Physics > Rigidbody*. This will let you set rules for how MazeRobo behaves in the game.
- You'll see now that when you have MazeRobo selected, there's a *Rigidbody* section in the Inspector. Open up the *Constraints* subsection

of the *Rigidbody* section, and set *Freeze Rotation X, Y and Z* to *True* by clicking in all the boxes. In *Freeze Position*, set *Y* to *True* by clicking that box.
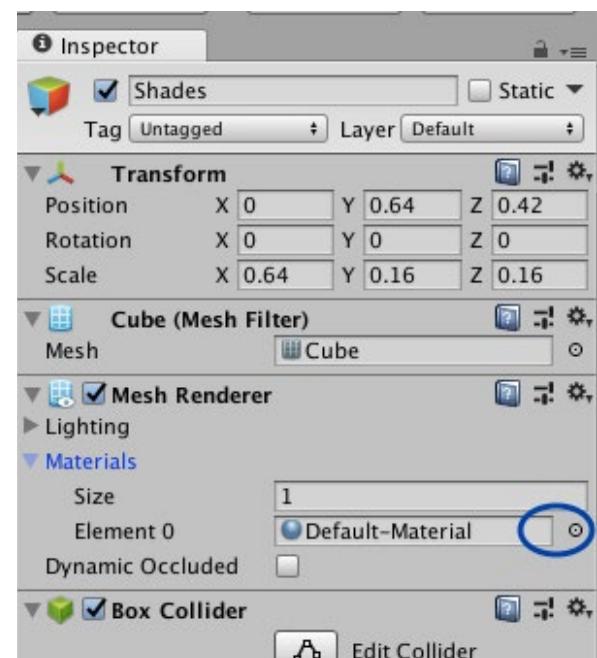
- Now you have a basic robot character you can use in your game. You can really make it your own by changing a few colours around, or maybe adding extra pieces to it using more *3D Objects* that you can position, like you did in the last step! Once you're happy with your robot, you can move on to the next step.
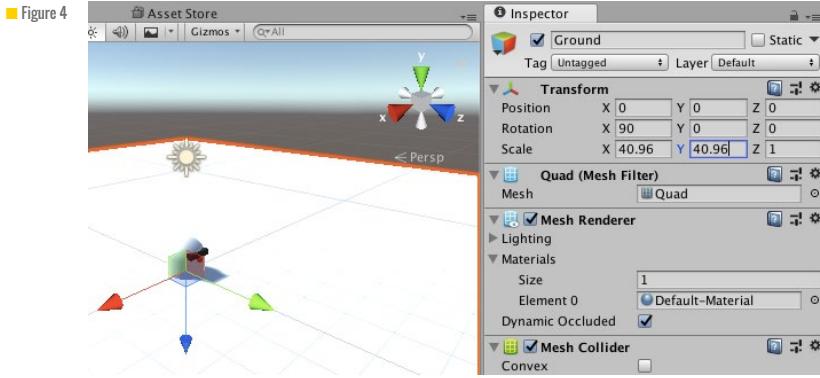
Make the world's ground

Now you're going to create a ground plane for MazeRobo to move about on:

- Start by adding a *Quad* object to be the ground (*GameObject > 3D Object > Quad*). Change the name of this object from `Quad` to `Ground` in the Inspector. ▶

■ Figure 4

■ In the Inspector for this new object under *Transform*, set the *X Rotation* to 90, and for *Scale*, enter these values:

```
X = 40.96
Y = 40.96
Z = 1
```

Gah! MazeRobo's stuck halfway into the ground! Let's move it up by one metre:

■ Select MazeRobo and, in the Inspector under *Transform*, set the following *Position* coordinates:
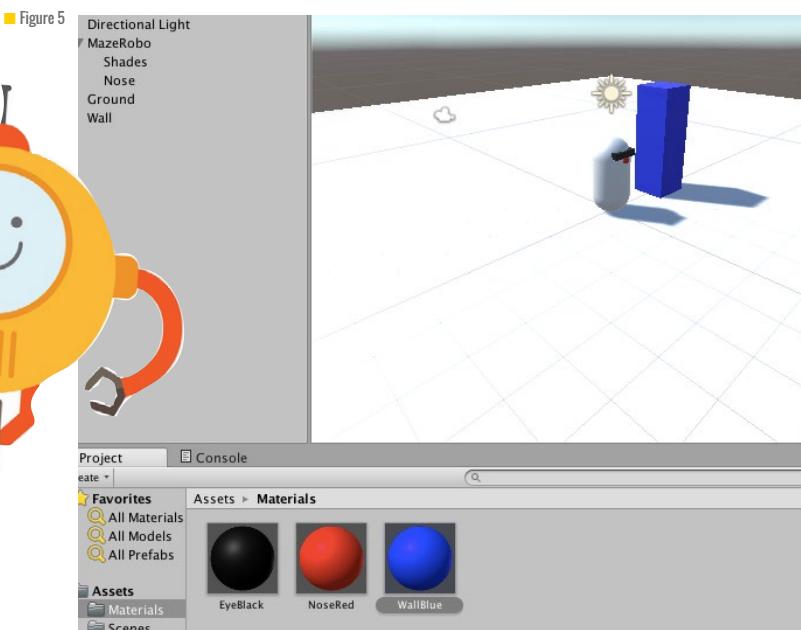
```
X = 0
Y = 1
Z = 0
```

Now you'll add a wall to start your maze:

■ Create a Cube (*GameObject > 3D Object > Cube*) and set its *Transform Position* to:

```
X = -2
Y = 1.5
Z = 0
```

■ Set the Y *Scale* to 3 and rename the object to `Wall`.
■ Now make a new material for `Wall` (*Assets > Create > Materials*), rename it `WallBlue`, and change its albedo to give it a (surprise!) blue colour.
■ Assign the `WallBlue` material to `Wall` using the *MeshRender > Materials* section of the Inspector (you can also drag the material straight onto the object).

Later on, you'll turn this wall into a maze for MazeRobo to explore! [Figure 5]

Make MazeRobo move

Time to write some code so that the player of your game can control MazeRobo:

■ Go to the Project pane and create a new folder (*Assets > Create > Folder*) inside the `Assets` folder (you may need to click on the `Assets` folder first). Call the new folder `Scripts`.
■ Create a new C# script (*Assets > Create > C# Script*) in this folder, and call it `RoboMover`.
■ Double-click on `RoboMover` to open it in an editor (which is a separate program from Unity). You should see code like this:

```
using System.Collections;
usingSystem.Collections.Generic;
using UnityEngine;

public class RoboMover :
 MonoBehaviour {

 //Usethisforinitialization
  void Start () {

  }

 //Updateiscalledonceperframe
  void Update () {

  }
}
```

Now that you've got some code to work with, it's time to start adding to it:

■ First, you need to add some *variables*, inside the class but before the functions, like this:

```
public class RoboMover :
MonoBehaviour {

 publicfloatmoveSpeed=4.0f;
  public Rigidbody rb;
  public Transform tf;
```


■ Figure 5

```
//Use this for initialization
void Start () {
```

- You don't actually need the `Start` function in this program, so you can delete these lines:

```
//Use this for initialization
void Start () {

}
```

- Now you need to fill in the `Update` function with this code:

```
//Update is called once per frame
void Update () {
   Vector3 desiredDirection
 = new Vector3 (Input.GetAxis
 ("Horizontal"), 0.0f, Input.
GetAxis ("Vertical"));
 desiredDirection=moveSpeed *
 desiredDirection;
   desiredDirection = Time.
deltaTime * desiredDirection;
 rb.MovePosition(rb.position+
desiredDirection);
 }
```

- Be sure to save your code (*File > Save*).

That should do it! You're close to getting MazeRobo moving now:

- Back in Unity, drag and drop your `RoboMover` script from the `Scripts` folder and onto the `MazeRobo` *GameObject* in the *Hierarchy*.
- You'll see that a field for the script is now visible in MazeRobo's Inspector, below *Rigidbody*.
- There are two empty fields in the `RoboMover` field: `rb` and `tf`. As you know, these stand for *Rigidbody* and *Transform*, and if you click and drag the names of these components from their places in the Inspector and into their respective fields, `RoboMover`
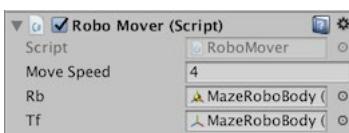
■ Figure 6
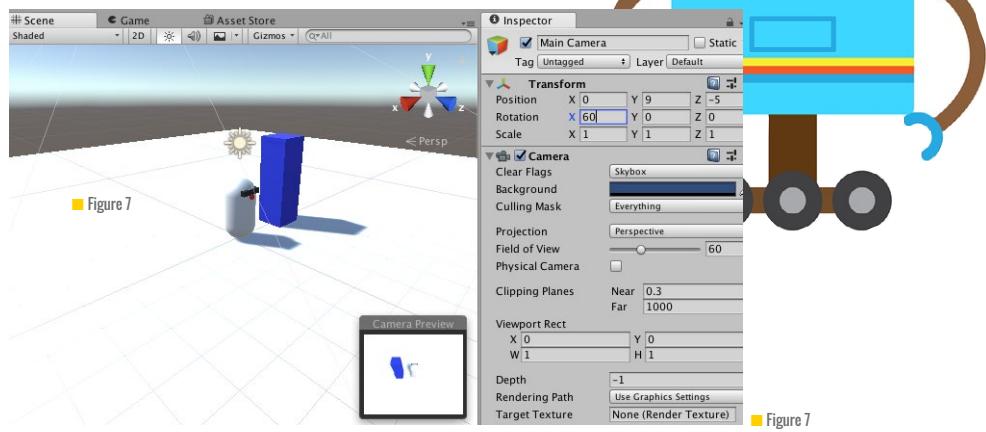
■ Figure 7

■ Figure 7

(the script) will have all the info it needs to move MazeRobo! [Figure 6]
- Now click on the big *Play* button at the top centre of the Unity interface...

MazeRobo moves!
- Use the arrow keys to control MazeRobo. When you're done, press the Play button again to stop the game.

Moving properly

MazeRobo moves, but it's a little... weird. It doesn't turn around to see where it's going. You can fix that:

- Go back into the `RoboMover` script and add this new line below `rb.MovePosition`:

```
rb.MoveRotation (Quaternion.
LookRotation(desiredDirection,
Vector3.up));
```

This line makes MazeRobo look where it's going.

- Run the game and check it out!

MazeRobo does look where it's going, but as soon as you release the controls it springs back to looking in its original direction. You can fix that, too:

- The first thing you'll need to do is wrap all your existing direction change code in an `if` statement, which only runs the code inside it if the condition in the brackets is `true`.

```
// Update is called once per
```

```
frame
void Update () {

   if (true) {
      Vector3 desiredDirection
= new Vector3 (Input.GetAxis
("Horizontal"), 0.0f, Input.
GetAxis ("Vertical"));
      desiredDirection =
moveSpeed * desiredDirection;
      desiredDirection = Time.
deltaTime * desiredDirection;
      rb.MovePosition (rb.
position + desiredDirection);
      rb.MoveRotation
(Quaternion.LookRotation
(desiredDirection,Vector3.up));
   }
}
```

- Run the game and check it's all still working.

Now create your test conditions:
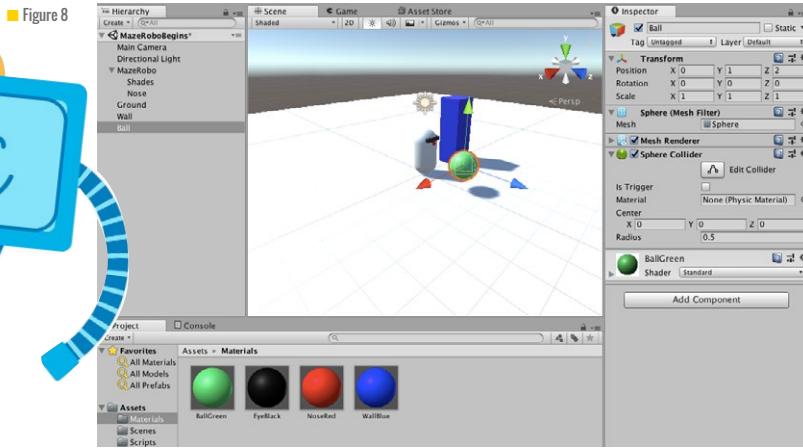
- Above the `if` statement but still inside the `Update` function, you'll need to collect the player inputs and get their *absolute* values like this:

```
void Update () {

float inputHorizontal=Mathf.Abs
(Input.GetAxis("Horizontal"));
float inputVertical=Mathf.Abs
(Input.GetAxis ("Vertical"));

if (true) {
```

Now it's time to update the `if` statement so it actually tests something! You'll

need to change what's in the brackets after the `if` so that it checks if `inputHorizontal` is *greater* than `0.01` or if `inputVertical` is *greater* than `0.01`, and gives a `true` result in either case.

To do this, you'll need to use an 'or' between your conditions that your computer can understand. In C# (the language you're writing your Unity scripts in), we represent 'or' with two pipe characters, like this: `condition A || condition B`. There are also other ways of joining two or more conditions, for example the 'and' operator (`&&`), and you can look those up online if you need them.

- To write the 'or' condition you need, update your `if` statement like this:

```
if(inputHorizontal>0.01f||
inputVertical > 0.01f) {
```

Now MazeRobo should stay facing the direction it's just moved in! If you're having any problems, check that your `Update` function matches this code:

```
void Update () {

   float inputHorizontal =
Mathf.Abs (Input.GetAxis
("Horizontal"));
   floatinputVertical=Mathf.Abs
(Input.GetAxis("Vertical"));

   if(inputHorizontal>0.01f||
inputVertical > 0.01f) {
      Vector3 desiredDirection
```

```
= new Vector3 (Input.GetAxis
("Horizontal"), 0.0f, Input.
GetAxis ("Vertical"));
 desiredDirection=moveSpeed
* desiredDirection;
   desiredDirection = Time.
deltaTime*desiredDirection;
 rb.MovePosition(rb.position
+ desiredDirection);
 rb.MoveRotation(Quaternion.
LookRotation(desiredDirection,
Vector3.up));
   }
```

Camera tracking

MazeRobo moves, but right now the camera always stays in the same place. That's going to be a problem if you try to add anything outside of the camera's initial field of vision, or anything that moves outside it (like MazeRobo itself!). Let's make it adjust:

- Select the *Main Camera* in the Hierarchy and set its *Transform* properties in the Inspector as:

Position
X: 0
Y: 9
Z: -5
Rotation
X: 60
Y: 0
Z: 0
Scale
X: 1
Y: 1
Z: 1

[Figure 7]

Now you've changed the camera's angle (run the game to test it if you like!), but it still doesn't follow MazeRobo. To make that happen, you'll have to update the camera's location every frame, and for that, you'll need another script:

- Create a new script (*Assets > Create > C# Script*) and call it `CameraMover`. Put it in the `Scripts` folder.
- At the start of the script, just inside the `CameraMover` *class*, add *three variables* like so:

```
public class CameraMover :
MonoBehaviour {

  public Transform tf;
  public Transform
playerTransform;
  public Vector3
distanceBetweenPlayerAndCam;
```

- Now you need to set the initial distance between MazeRobo and the camera as the one you want to keep. Do this inside the `Start` function like so:

```
void Start () {
 distanceBetweenPlayerAndCam=
tf.position-playerTransform.
position;
 }
```

- Next, ensure that the game keeps that distance the same in every frame of the game by adding a line to the `Update` function like so:

```
void Update () {
  tf.position =
playerTransform.position +
distanceBetweenPlayerAndCam;
 }
```

- You need to attach the script to the camera now, so go back to Unity and select the Main Camera in the Hierarchy. Then drag the `CameraMover` script from the Project space onto the Main Camera.
- Find the *CameraMover* field in the Inspector, and drag the Main

Camera from the Hierarchy into the *Tf* field. Then drag MazeRobo from the Hierarchy into the *Player Transform* field.
- Now run the game and watch the camera follow MazeRobo around!

Give MazeRobo something to play with

Now that you've got your character and it's moving around properly, it's time to give it something to play with. You're going to add a ball for the MazeRobo to push around:

- Start by creating a Sphere (*GameObject > 3D Object > Sphere*). Rename it `Ball`.
- Set the *Transform Position* property of `Ball` to:

```
X: 0
Y: 1
Z: 2
```

- Make another material (see the *Add colour* step if you don't remember how!) and make it green. Call it `BallGreen` and drag it onto `Ball`. [Figure 8]

If you test the game now and walk MazeRobo into the sphere, you'll notice it behaves just like the wall: it doesn't move. You want the sphere to be a rolling ball, though, so you'll need to give it some rules for moving, like MazeRobo has:

- Select `Ball` and give it a *Rigidbody* component (*Component > Physics > Rigidbody*).
- Now try playing!

Winning!

You've got a robot, you've got a ball… now in order for it to be a game, there's got to be a way to win. You'll be adding that now:

- First, add another cube and call it `WinZone`. Maybe give it a new, noticeable colour (yellow? orange? pink?).
- Make sure you have `WinZone`

selected in the Hierarchy, and in the Inspector under *Box Collider*, select the *Is Trigger* option.
- Set the *Transform Position* property of `WinZone` so that it's:

```
X: -5
Y: 1
Z: -2
```

You're going to write another script to let `WinZone` detect when the `Ball` touches it. In order to do that, the ball needs to be *tagged*:

- Select `Ball` in the Hierarchy, and in the *Inspector* select the *Tag* field just under its name.
- Choose *Add Tag…*, then click on the + icon and create the tag 'Ball'.
- Reselect `Ball` in the Hierarchy, select the *Tag* field again, and choose the 'Ball' tag you just created.

While you're at it, why not add some celebration to let the player know when they've won?

- Create a *Particle System* (*GameObject > Effects > Particle System*) and call it `Fireworks`.
- Select the `Fireworks` object and unselect the box beside its name in the Inspector. This hides the object, so you can make it appear once you're ready to set off the fireworks!
- Now look in the list of settings in the Inspector, find *Start Color* and set it to yellow, or green, or whatever you like really!
- Finally, make the *Position* of the `Fireworks` match the *Position* of `WinZone`.

Now you'll add code to make the fireworks appear at the right time:

- Create a C# script (in the `Scripts` folder) called `WinZone`. Open the new script and remove the `Start` and `Update` functions. Put this code inside it instead:

```
public GameObject fireworks;
```
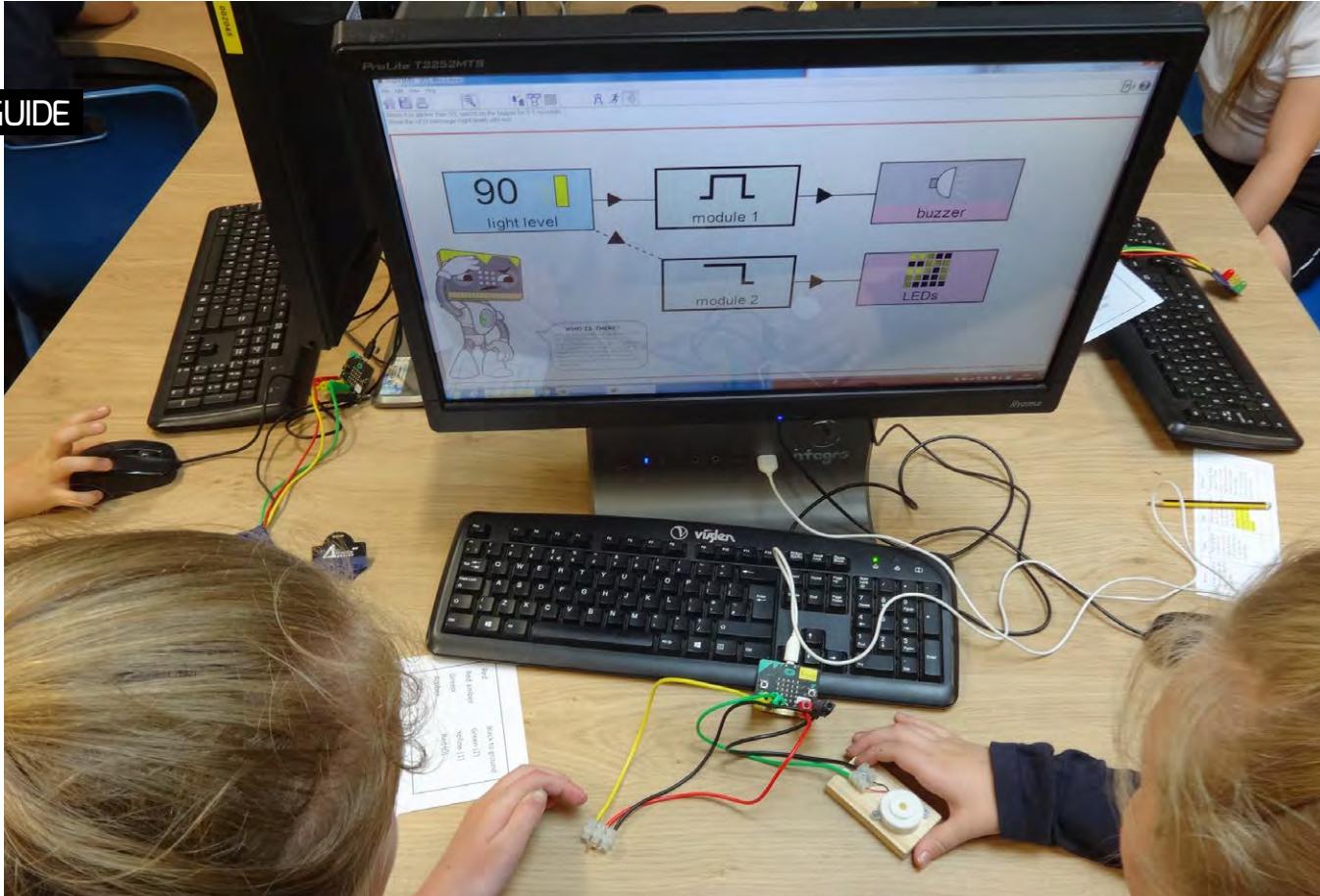
■ Figure 9

```
void OnTriggerEnter (Collider
col) {
  if (col.transform.CompareTag
("Ball")) {
    fireworks.SetActive (true);
  }
}
```

- Save the changes to the script and go back into Unity.
- Drag the script onto the `WinZone` in the Hierarchy and then, with `WinZone` selected, drag the `Fireworks` object from the Hierarchy into the *Fireworks* field in the *WinZone* section of the Inspector.
- Save the game, run it, and put the `Ball` in the `WinZone`. See what happens! [Figure 9]

That's all the basic game pieces. (HW)

MICRO:BITS WITH MR BIT
AT RICHMOND PRIMARY SCHOOL

Getting started with programming the BBC micro:bit to control simple devices is much easier than you might think

GUIDE BY Dawn Cox

When the BBC micro:bit first came on to the scene, its built-in sensors and facilities for connecting even more external sensors and devices offered the promise of giving pupils a simple first-hand experience of computer control. This area of the computing curriculum has tended to be under-resourced in primary schools, so I saw a golden opportunity to plug this gap in our school.

After-school club

The first task was to obtain a class set of micro:bits. Fortunately, our regional CAS hub (Nottingham Trent University) was able to help us here with the loan of a set. I decided to introduce them first to a small

group of pupils in my after-school club. To begin with, we used iPads equipped with Insight Mr Bit software. I chose this software partly because it avoided the complication of needing internet access to code editors, and partly to test its 'Plain English' method of programming, which seemed to offer a very easy pathway to computer control. The iPad uses a Bluetooth wireless connection with a micro:bit, and this worked well for most pupils, but troubleshooting, when needed, was rather time consuming and disruptive of the flow of the session. We later decided to use the Windows version of the software on our suite of PCs, with USB connection to the micro:bits.

Working in class

Having gained confidence with micro:bits and Mr Bit in the after-school club, I scheduled to use them with two Year 5 classes. Then with 30 pupils at a time, I organised working in pairs for the hands-on activity at the computers. In our computer suite, we are fortunate to have a central carpeted area where the children can be quickly assembled to sit in front of a large screen, ideal for whole class teaching and discussion.

Programming in 'Plain English'

The built-in tutorials and exercises in Mr Bit made it very easy to organise a graded sequence of lesson activities. These can

be easily differentiated. All the instructions are in resizable speech bubbles on the screen. Children soon pick up the input/output concept from the visual blocks, and I encourage them all to check that the English sentence program scripts, at the top of the screen, describe what they are aiming for the program to actually do. These scripts are the key to the 'Plain English' method; pupils link up the inputs and outputs, decide on the starting and finishing conditions, and the program builds the sentences based on their choices.

Controlling devices with sensors

When it came to connecting sensors and devices to the micro:bits, we used a simple kit designed by Dr Laurence Rogers from Leicester University: a temperature sensor, light sensor, buzzer, and a set of traffic light LEDs.

Fitted with colour-coded 4mm banana plug leads, the children had no difficulty in connecting the simple circuits. Each of the scripted activities with the kit featured an everyday application of the sensors, so the context of embedded computers in everyday life was ever present. In class discussion, we thought about mobile phones, washing machines, TV sets, photocopiers, and so on, discovering that so many of these devices were controlled by embedded computers containing

programs of the sort we were creating in class.
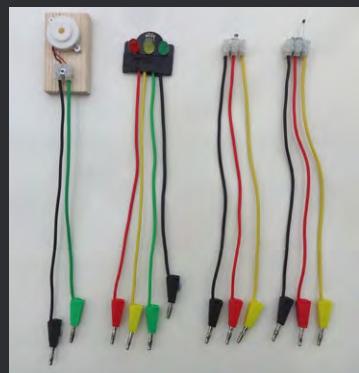
On the large screen, pupils demonstrated their solutions to their peers, explaining the steps that they took as per the Mr Bit instructions. Sometimes they would adapt their program to work in a slightly different way, or they would devise an equivalent solution using their own ideas.

As pupils' own ideas emerged, I realised that there were plenty of opportunities for differentiating pupils' performance, and I began to set differentiated tasks which I could use for our target tracker assessments related to the National Curriculum. At the lowest level, I would expect pupils to create a program script by following the Mr Bit step-by-step instructions. At higher levels, I would expect pupils to adapt the program, or create a program without the instructions, simply responding to a brief explaining the goal. Amongst the teachers' resources freely available from the Mr Bit website, I gave each pupil a progress card to chart their path through the activities, and awarded certificates to recognise achievement.

Mr Bit is fun

Using the Mr Bit software and activities has been fun and engaging for pupils of both sexes. The scripted activities are visually attractive and have become a popular voluntary activity during lunchtimes. Even

DIY SENSORS AND DEVICES
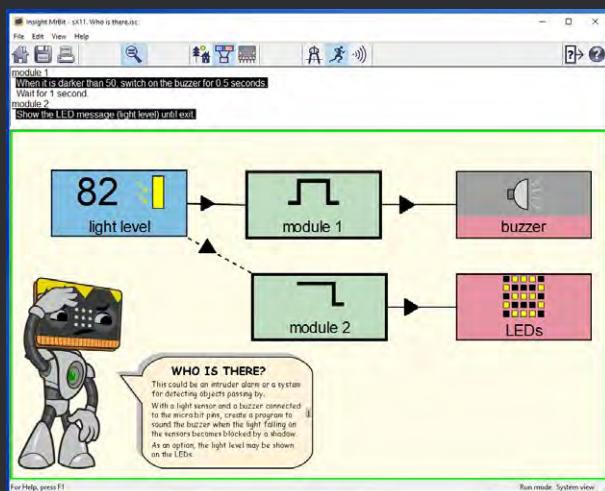
You can make a simple kit for plugging into the micro:bit with inexpensive components from the usual suppliers: temperature and light sensors, buzzer, traffic light LEDs, terminal connectors, and 4mm plug leads. No soldering needed - just use wire cutters and a screwdriver. The stackable 4mm plugs are robust and give reliable connections to the micro:bit. Details at www.insight-mrbit.com

Year 2 pupils have been seen dipping into the resources. From a teacher's point of view, the huge range of activities is of exceptional value and ticks many boxes in our target-focused computing curriculum.

Find out about Mr Bit resources at **www.insight-mrbit.com**. (HW)

Dawn is Computing Coordinator at Richmond Primary School, Hinckley, where she has taught computing for over a decade. She is passionate about computing and takes great pleasure in watching her pupils' skills develop, often far above her initial high expectations. She believes this success is due to being part of an extremely supportive team of staff, and having opportunities to further her knowledge and understanding of new programmes and technologies through CAS hub meetings and conferences. She is a great believer in collaboration between teachers and schools and, within the TELA group of primary schools in Leicestershire, has organised training sessions and meetings to promote the sharing of resources.

PROGRAMMING IN 'PLAIN ENGLISH'

The Mr Bit screen shows input and output blocks linked to one or more control modules. The program for each module appears in the script at the top of the screen. As pupils link up the inputs and outputs, and decide on the starting and finishing conditions, the program builds the sentences based on their choices.

ONE DESIGN, THREE WAYS

We are required to teach children to both 'design, write and debug programs' and 'select, use and combine a variety of software on a range of digital devices to design and create a range of programs, systems and content that accomplish given goals'. Is this really achievable?

STORY BY Matthew Wimpenny-Smith and Jane Waite

Why are design and implementation important?
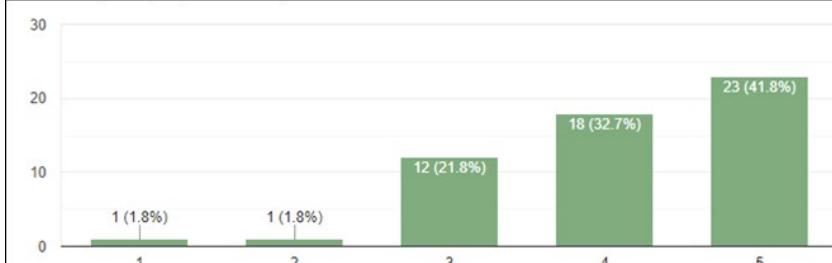
Design is very important and can be often overlooked in computing projects, as we tend to be too quick and just jump straight into the coding or making stage. Without good design and an understanding of how to implement it, we don't get good products, services, or code for that matter! So why do we encourage children to be creators rather than consumers but without design?

Back in April 2018, I attended a fantastic three-day course run by Jane Waite called Diving Deep into Primary Programming, where there was much discussion and debate about how to implement design in computing lessons. This academic year, I've set myself the goal of getting my pupils to really think about design and how to implement it.
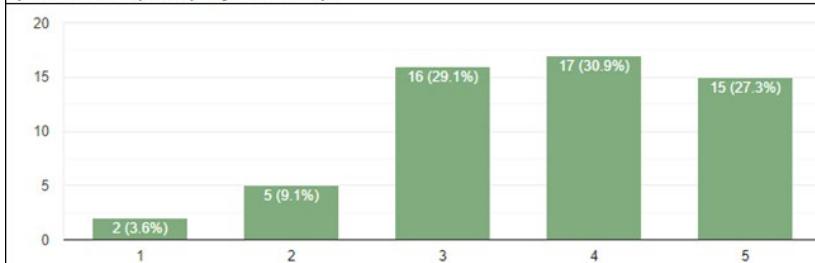
Of course, it also requires a shift in the mindset of both the pupils and myself to spend time thinking about design alongside the correct pedagogy to suit. This includes helping them not to be tempted to just jump in and start coding, but to critically think and apply computational thinking skills. To learn to know their own limitations, such as what is possible within the software and hardware they're using, how long they have for a project, and what skills they'll need to meet their design ideas – what is 'doable'.

I'm very fortunate to teach computing across all year groups in my school, and over the last six years we've done a bit of design, but usually not with a clear purpose of what I was expecting, or how we would be using the designs within

the coding process. I therefore wanted to formalise this, having noticed that spending time on design helps with the process of decomposition and abstraction skills of computational thinking. Alongside this, we need to help pupils evaluate software and hardware, thinking about

what is possible, and decide what might be the best way to implement a design.

Christmas message project

It was a sunny October day in London and, as I was having lunch with Jane Waite, the conversation turned to what I

How confident do you feel now about making your Christmas message in Slides on a scale 1 (not confident) to 5 (very confident)?

How confident do you feel now about making your Christmas message in Scratch on a scale 1 (not confident) to 5 (very confident)?

How confident do you feel now about making your Christmas message using the Micro:bit on a scale 1 (not confident) to 5 (very confident)?

■ Results of Google Form pre-assessment show confidence within the three systems

All images courtesy of Matthew Wimpenny-Smith

was planning to teach my Year 6 pupils after half term. I said, "I am thinking about starting to transition them from blocks to text coding". Jane asked "Are they ready to move on?" and warned me not to do it too soon, empathising the need for a firm foundation of block-based programming. I was intrigued to find out, so over that lunch meeting we thrashed out the idea for a design implementation project that would give me more information on what they currently knew and could do.

The project was simple: could the pupils create a design and independently implement it in a variety of software that I felt they were proficient in, including block-based code, and then could I move them on to a text-based language using the same design?

Christmas was approaching, so we settled on the idea of creating a Christmas message for other pupils in the school, with the added Modern Foreign Language twist of it being in different spoken languages, such as French, Japanese, and so on.

I chose Google Slides, Scratch, and a micro:bit for the implementation, then started with a pre-assessment to ascertain the pupils' confidence within the three implementations. The results of this assessment showed without a doubt that they felt most confident using Google Slides, followed by Scratch, and then the micro:bit, as highlighted by the bar graphs.

The results of this are, of course, predictable, mainly due to the pupils having lots of exposure to Google Slides across all other lessons, and limited exposure to Scratch, mainly via my computing lessons (one hour per week plus one hour of Code Club for those who were keen). And only those who attended my Code Club, had enthusiastic parents, or siblings in the senior school had any exposure to the micro:bit prior to this project. I did, however, briefly demonstrate the micro:bit just before the pre-assessment.

Design stage

After the pre-assessment, the pupils worked on their paper designs, followed by a self-assessment for confidence using red, amber, green (RAG). The challenge set was that they needed to implement their design as closely as possible within the three

■ This image shows the pupil design being implemented in Google Slides
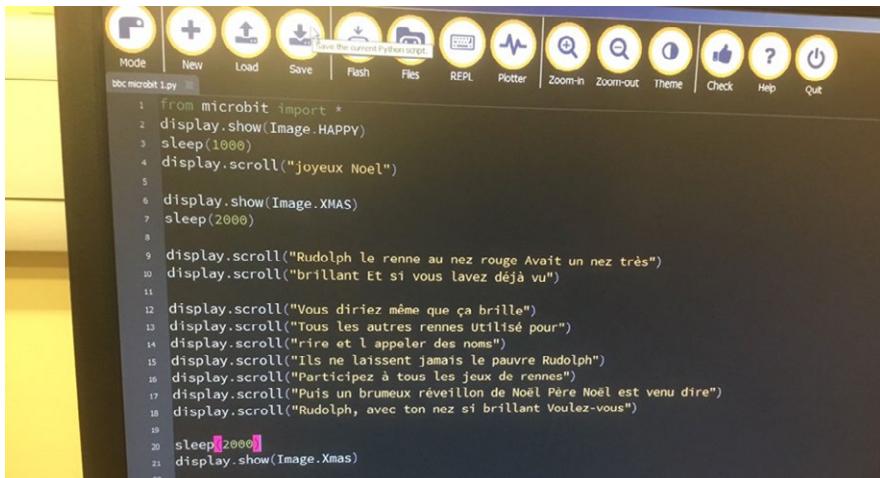

■ This image shows the pupil design being reviewed based on the implementation

different systems (Slides, Scratch, and micro:bit). Between each implementation, I photocopied their design and asked them to re-assess it for confidence, and also make notes as to how they would need to change their design to fit the limitations of the software. This helped to highlight their initially ambitious ideas and hone their critical thinking to what is actually possible given their knowledge and understanding, plus the curriculum time constraints.

Implementation stage
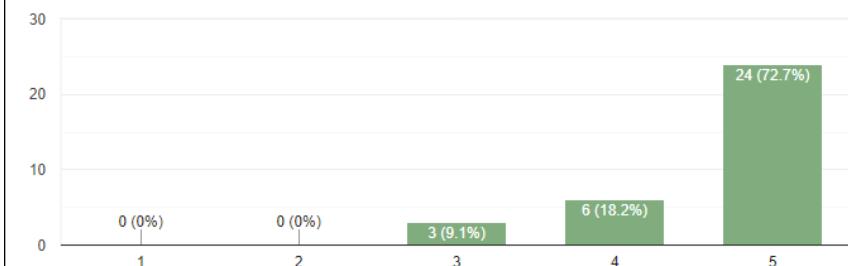
The pupils found that they could implement their designs with little change in both Slides and Scratch, drawing

on their existing knowledge; here is a link to the implementation in Scratch: **helloworld.cc/2LtaXrb**. However, there was realisation after pre-teaching using the micro:bit that this was a different approach, moving away from block-based language to a text-based interface, and that their designs would need to be modified. I purposefully kept them constrained to just being able to display and scroll text and images on the micro:bit LED matrix, and I would use the text-based Mu editor. The next three lessons were a busy time for Year 6 as they adapted their design and taught themselves how to scroll and display ▶

```
from microbit import *
display.show(Image.HAPPY)
sleep(1000)
display.scroll("joyeux Noel")

display.show(Image.XMAS)
sleep(2000)

display.scroll("Rudolph le renne au nez rouge Avait un nez très")
display.scroll("brillant Et si vous lavez déjà vu")

display.scroll("Vous diriez même que ça brille")
display.scroll("Tous les autres rennes Utilisé pour")
display.scroll("rire et l appeler des noms")
display.scroll("Ils ne laissent jamais le pauvre Rudolph")
display.scroll("Participez à tous les jeux de rennes")
display.scroll("Puis un brumeux réveillon de Noël Père Noël est venu dire")
display.scroll("Rudolph, avec ton nez si brillant Voulez-vous")

sleep(2000)
display.show(Image.Xmas)
```

■ This image shows the implementation of the design in the Mu editor
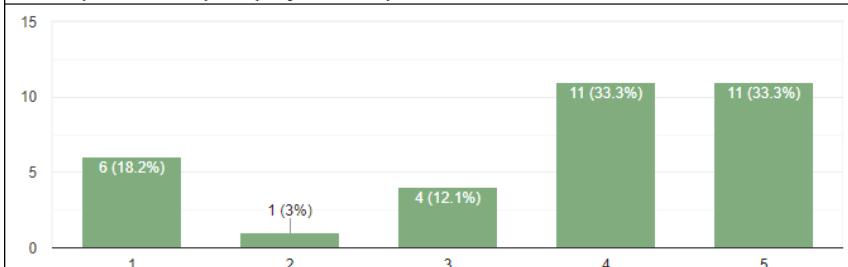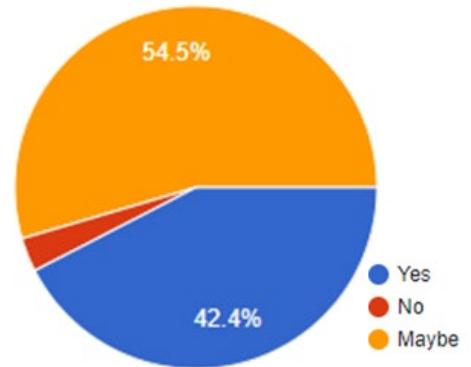
■ Summary of the post-assessment survey results

■ Pie chart showing results of asking the pupils
if they are ready to move on from Scratch

▶ images on the micro:bit. They also discovered that there were limitations, such as not being able to display certain languages, especially Cyrillic-based alphabets – they would just scroll '?'s.

Post-assessment

The post-assessment survey highlighted some interesting results. Predictably, the pupils' confidence with using Google Slides was as I expected, but interestingly there was a shift in how they felt about Scratch. It must be noted that for this project, I intentionally gave limited support with both Slides and Scratch.

The results and comments from the post-assessment suggested that pupils found following a design constraint in Scratch harder than they initially thought. This has led me to realise that they overestimated their abilities within Scratch. One pupil finding it challenging remarked, "please can I just do the whole project in Slides!"

Perhaps one reason why the pupils may have been overconfident with Scratch is that in the past they've tended not to stick to an initial idea once they've got started, and to just do what was easy as they wrote the code, or they're used to simply copying code without really understanding it. Scratch is complicated, and working out how to implement a design isn't easy. I realise I need to work more on this, to show them how to implement ideas and know what is doable. It seems that the current way I teach Scratch isn't quite hitting the mark. I need to do more research in my class to explore this.

This was empathised by the results of another question asking them if they felt ready to move on from Scratch. The results showed that more than 50% felt they were not sure or 'no' they weren't ready. Based on this classroom research, it has highlighted the need for me to ask my pupils about their learning more often, and not to move them on until they're truly ready. Finally, I need to do more to develop my pedagogy of teaching computer science. (HW)

DIRECTLY DRIVE LEARNERS' UNDERSTANDING

Make use of the direct drive principle in whatever programming resource you're using

STORY BY Julia Briggs

ScratchMaths materials were developed by UCL Institute of Education to identify ways in which the language of programming can help to support the development of mathematical understanding.

The modules, available from **helloworld.cc/2ST7a9a,** are based on some 'Big Ideas'. The big, and yet simple, idea of direct drive has been used by Somerset's eLIM team to provide a guided exploration approach to build confidence and understanding in both learners and teachers.

A progression of six steps is hypothesized, from playing with a toy to programming a sprite in Scratch:

- Directly manipulate a physical toy

- Directly drive a toy by pressing a button

- Build future behaviour of a toy – programming

- Directly manipulate a sprite (for example. by dragging)

- Directly drive a sprite by giving it a command

- Build future behaviour of a sprite – programming

Direct drive is important for children as it provides a foundation for them to clearly see a single action having a single visible effect. A sprite in Scratch can be manipulated by dragging it around the stage area of the

■ What happens if you click on individual movement blocks?

software. You can drive the sprite by clicking on a single programming block and causing something to happen. You'll see a 'single, immediate, visible, and unambiguous effect'. This learning is then used to plan future behaviour of the sprite as a sequence of programming blocks is put together.

I remember the excitement of two girls I observed during research in 2013. They selected three of the movement blocks and, clicking on them individually, discovered ways to move and turn their character around the screen. Their enjoyment was clear as they giggled together over what they made happen. Their discussion showed how they were independently and collaboratively developing understanding.

A single movement block, or sensing block, can be investigated. What is the

■ The five-block challenge

block doing? What will the block allow us to make happen?

A five-block challenge will allow children to investigate blocks before building the behaviour they want for a planned outcome. The rules are:

- Only these blocks can be used

- Begin by clicking on blocks individually before creating a sequence

- Blocks can be used more than once, and you don't have to use them all

- Numbers or text in a block can be changed

For some contexts, a three-block challenge might be more appropriate, for others a 10-block challenge. A teacher can set the challenge based on the learning they're planning. A new block can be introduced to prepare the children for an activity.

The eLIM team was fortunate to act as a hub for ScratchMaths research in 2016 and 2017, and continue to be excited by the way in which direct drive scaffolds learning when using any programming resource. We were impressed with the sheer enjoyment of the materials overall by both learners and teachers. Select the blocks of work that fit with your computing and maths planning. Consider use of Module 1 in Year 4. **(HW)**

Julia is a part of Somerset's SSE eLIM team. With thanks to Professor Ivan Kalas, Piers Saunders, and Dr Laura Benton.

DIETING WITH
GENETIC ALGORITHMS

For a simple calorie controlled diet, we'd like to eat as much as possible, subject to a maximum calorie count. Counting calories may be good for mental arithmetic, but it's hard to keep coming up with new recipes. So, how about automatically generating possible meals?

STORY BY Greg Michaelson

Encoding and fitness

Planning a diet is an example of a *knapsack problem*. Given a set of items with different properties, the goal is to come up with a collection of items that maximises some properties, say weight of food, subject to restrictions on others, say number of calories. A simple approach is to systematically search all possible combinations of items, but the time required grows very quickly with the number of items available. Instead we're going to look at using a *genetic algorithm (GA)* to evolve food combinations that meet our requirements.

As the name suggests, GAs are motivated by biological evolution – see box. From a computing perspective, the key idea is that we can encode a solution to a problem as a sequence of values. We then generate possible solutions, assess them, and change them to try to nudge them nearer to an ideal solution.

For our problem, let's start with a table of foodstuffs and calorie counts. Figure 1 shows 12 food items, with kilo calories per 100 grams, arranged along rows by increasing calorie count.

We could encode a meal as a sequence of 12 integers, for the weights of the corresponding food items. For example, for a stir fry we might have:

100 grams mushrooms, 100 grams green beans, 100 grams red pepper, 100 grams onion, 100 grams chicken, 10 grams garlic, 10 grams olive oil →

food	Kcal / 100 g	food	Kcal / 100g	food	Kcal / 100g	food	Kcal / 100g
mushrooms (M)	18	tomato (T)	20	green-beans (GB)	25	red-pepper (RP)	30
carrot (C)	34	onion (O)	37	potato (P)	80	chicken (Ch)	105
garlic (G)	106	egg (E)	155	lentils (L)	319	olive-oil (OO)	823

■ Figure 1: Food and calorie counts

100,0,100,100,0,100,0,100,10,0,0,10

In a GA, we assess a solution in terms of how well it meets some criteria, that is how fit it is. Typically, this involves some calculation using the values of the solution elements.

Suppose our diet allows 250 kilo calories for dinner. Then we'll assess the fitness of a meal as the absolute difference between its total calories and the 250 kilo calorie ideal. For our stir fry, we have:

(100*18+0*20+100*25+…10*823)/100 = 307 kilo calories

So the fitness is |307-250| = 57.

How genetic algorithms work

Given an encoding and a way of assessing fitness, we run a GA against a *population* of possible solutions to find the fittest one.

We'll look at the general technique through the purely illustrative example shown in Figure 2. Suppose we have a suite of colours of different values, and we're looking for any combination of five colours with a total value 9. So the fitness will be |9-sum of colour values|.

A GA starts with a random *initial population* (a). The GA then calculates the *fitness* of each solution against some criteria (b) and rank orders the solutions by fitness (c).

Of course, to start with, most solutions will be very unfit. Nonetheless, some will be better than others. So the GA will reject the worst solutions and replace them with copies of the better ones; that is, it will *reproduce* the better solutions at the expense of the worse ones (d).

If a GA did nothing else to the population, and repeated this process for long enough, eventually it would end up with a population entirely composed of the original fittest solution. So, a GA introduces change into the population through:

■ *Mutation*: make random changes to random elements of random solutions (e)

■ *Cross over*: swap random elements of pairs of random solutions (f)

Thus, a GA continues through successive iterations of these stages until a best

solution is found. See Figure 3 overleaf for a summary.

This all sounds entirely haphazard, but, as we'll see, a GA can quickly prune the search space of a knapsack problem. The worst that can happen is that it will get stuck on a plateau where solutions never get any better. However, given the high degree of randomness, each run of a GA is likely to be different. Furthermore, parameters for population size, reproduction rate, mutation rate, and cross over rate can be tuned.

A GA for meals

For our problem, let's work with a population of 500 possible meals. We could hold these in a 100 * 13 array of integers, where each row is a candidate solution consisting of 12 food element weights and the fitness.

Our GA will:

- Generate 500 initial solutions, each with 12 random weights, say between 0 and 999 grams

- Calculate the fitness of each solution and sort them in ascending order of fitness, because a low fitness value, i.e. close to 250 kilo calories, is desirable

- Reproduce the best 50%, that is copy the top 250 solutions over the bottom 250 ▶

■ Figure 2: Genetic algorithm actions

Figure 3. Genetic algorithm stages

iteration	M	T	GB	RP	C	O	P	Ch	G	E	L	OO	fitness
1	701	513	969	655	430	664	294	129	386	136	4	92	2565
3	701	513	39	655	430	664	294	129	386	136	4	92	2332
5	701	513	39	655	430	664	294	129	85	136	4	92	2013
7	701	513	39	655	430	136	173	129	4	136	4	92	1635
9	701	513	39	121	430	136	173	129	4	136	4	92	1475
11	701	136	39	121	430	136	173	129	4	136	4	92	1400
13	173	513	39	946	257	136	294	129	4	136	4	4	940
15	173	513	39	4	4	136	294	129	4	136	4	4	572
17	173	513	39	4	4	136	294	129	4	39	4	4	422
19	173	513	39	4	4	136	84	129	4	39	4	4	254
21	173	513	39	4	4	136	4	129	4	39	4	4	190
23	173	513	39	4	4	136	4	4	4	39	4	4	59
25	173	129	39	4	4	27	84	4	4	39	4	4	5
29	173	129	39	4	4	4	84	4	4	39	4	4	3
33	173	129	4	20	4	27	84	4	4	39	4	4	2
40	173	129	4	20	4	20	84	4	4	39	4	4	0

Figure 4: Towards a 250 kilo calorie meal

food	gm	Kcal/ 100 gm	KCal	food	gm	Kcal/ 100 gm	KCal
mushrooms	173	18	31	potato	84	80	67
tomato	129	20	25	chicken	4	105	4
green-beans	4	25	1	garlic	4	106	4
red-pepper	20	30	6	egg	39	155	60
carrot	4	34	1	lentils	4	319	12
onion	20	37	7	olive-oil	4	823	32

Figure 5: A 250 kilo calorie meal

- Mutate 5% of the population, that is choose 25 random solutions and change a random element to a random value between 0 and 999

- Cross over 10% of the population, that is swap random elements of each of 25 random pairs

Let's try running the GA for our problem. Figure 4 shows the best solution on every other of 40 iterations from a poor to a perfect solution, omitting iterations where the most fit doesn't change.

We can see that the weights of some items decrease quickly and stabilise, where others barely change. We can also see new numbers appearing, possibly through mutation, for example for garlic (G) before iteration 5. And we can see the same numbers recurring in different positions, possibly through cross over, for example garlic (G) getting the same value as lentils (L) before iteration 7. It's important, though, to remember that these are snapshots of fittest single meals from a population of 500 and we have no idea about the values in the whole population.

The fittest meal on this run is shown in Figure 5. Essentially, we are offered mushroom, tomato, potato, egg, and a little olive oil. A bit more egg, and we might make an omelette.

Conclusion

Here, we've only considered calories, but we could easily add maximising weight to the fitness. Then, the algorithm would gravitate towards meals with large amounts of low-calorie food, like mushrooms and tomatoes. And if we extend our characterisation of food, we can add more elaborate fitness requirements, like the balance of food types, and what goes with what, and our likes and dislikes, or even how often we want to eat something. In general, once we have an initial GA system working for a given encoding, we're free to change the data characterisation and the fitness criteria. (HW)

Greg Michaelson, Heriot Watt University.

MARK THORNBER TEACHER

MATHEMATICAL MUSINGS

The Shoelace Formula can determine the area of a simple polygon

C alculating areas takes up a lot of time in maths lessons. There are lots of formulas for different shapes, and sometimes more than one formula for the same shape, depending on the information known. There's a formula for area of a triangle if you know base and perpendicular height, and a different formula if you know all three sides, for example.

Life is a lot easier in computer graphics! All shapes on a computer screen are actually just polygons; even the apparent curves are made up of a finite number of pixels which can be thought of as vertices of a polygon with a lot of sides. In this case, one simple method always gives the area. It's known as the Shoelace Formula and goes as follows:
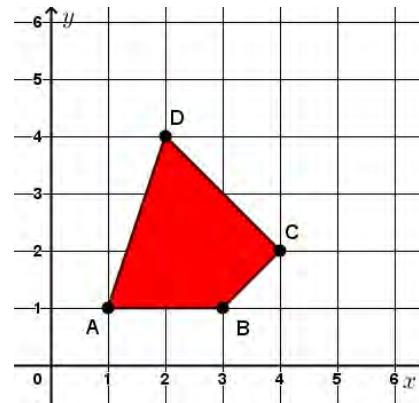
- List the vertices in order, going around the outside anticlockwise
- Make a table with x coordinates in the first column and y in the second
- Now multiply each x value by the y value in the row below, wrapping around at the end
- Multiply each y value by the x value in the row below, wrapping again
- Add the first set of numbers and subtract the second, now divide by 2 to get the area

As an example, consider the polygon pictured above.

This polygon gives the table of coordinates at the right. The calculation is shown by the two sets of arrows, hence the name "Shoelace"!

Area= ½(1/2) (1×1+3×2+4×4+2×1 - 1×3-1×4-2×2-4×1)

=½(1/2) (25-15)=5

Point	x	y
A	1	1
B	3	1
C	4	2
D	2	4

```
def polyArea(listOfCoordinates):
    c = listOfCoordinates
    l = len(listOfCoordinates)
    sum = 0
    for i in range(l):
        j = (i+1)%l  #The row "below"
        sum = sum + c[i][0]*c[j][1]
        sum = sum - c[i][1]*c[j][0]
    return sum/2
coordinates = [[1,1],[3,1],[4,2],[2,4]]
print(polyArea(coordinates))
```

As a matter of programming technique, it's better to add and subtract alternately as we move down the table. This avoids calculating two very large numbers and then subtracting, which could lead to overflow or rounding errors.

A small Python program to implement this is given above. The coordinates of a point are stored in a list, and then the shape is stored in a list of lists.

If you want to know how this works, there are some nice pictures at **helloworld.cc/2AdCdp8** (HW)

THE NEW NATIONAL CENTRE FOR COMPUTING EDUCATION

WHAT'S IT ALL ABOUT?

Everything you need to know about the new National Centre for Computing

STORY BY Sue Sentance

Hopefully, few readers (in England, anyway) will have missed the announcement from the DfE on 7 November that there is to be a new National Centre for Computing Education, with additional programmes to support GCSE computer science and A-Level computer science. This represents a massive investment by the government of £78 million, which is fantastic news for our subject. The National Centre and associated programmes will be run by STEM Learning, BCS, and Raspberry Pi.

Hello World, the magazine for computing teachers, will be keeping you up to date with all the developments with the National Centre. Here's a bit of a taster, but it's early days and there will be much more news in the next issue!

The National Centre will not be physically located anywhere. Instead, 40 school-led hubs will be created around the country, so if you're in England there will be one near you! Although led by a single school, a hub will be supported by an alliance of local schools and other relevant organisations, as well as central teams in the consortium (STEM Learning,

> ## THE NATIONAL CENTRE WILL EXIST UNTIL AT LEAST 2022, SO THERE'S MUCH MORE TO COME AND TO LOOK FORWARD TO!

BCS, and Raspberry Pi), in order that they can provide high-quality CPD, resources, and professional networks to primary and secondary teachers. We've had lots of enquiries about how schools are chosen to become a hub, and more details will be available on the new website, **teachcomputing.org**, which will be launched in January.

So what's available already?

The National Centre was launched in November with a single website page that will be replaced by a new website in January. Here you can sign up for more information and link to courses of interest.

If you teach GCSE computer science, or would like to, there's a unique programme just for you which is already currently available, called CS Accelerator. This programme includes at least 40 hours of face-to-face and online training, with an assessment at the end, to ensure that you feel confident to teach GCSE computer science (and have a certificate to prove

it!). Two-day face-to-face courses are available in York, with many more locations to be announced soon, covering Python programming, networking, algorithms, and data structures. We also have complementary online courses that you can take part in at your own pace, to consolidate and extend this learning. You can take as many of these as you like, and they're free for everyone, forever. Topics include: Python for Educators, Representing Data with Images and Sound, How the Internet works, Maths and Logic for Computing, Think Like a Programmer, and How Computers Work. The online courses have been developed by the Raspberry Pi Foundation, with support from Google, which has enabled some of these courses to be available already at the start of the programme. Bursaries are available for eligible teachers on the CS Accelerator programme.

There's also CPD available for primary teachers and teachers of KS3 computing. Online courses are available on Teaching Programming in Primary Schools, and Scratch to Python: Moving from Block- to Text-based Programming, with more to follow. Face-to-face CPD courses such as Primary Programming and Algorithms are also available already at the STEM Learning Centre in York, with more locations available shortly.

Also available is the CAS network of Communities (formerly called hubs), which exist to support you very locally. There are around 250 existing hubs, and you can find details of your local hub by signing up to CAS and entering in your location. These mostly meet termly and enable you to meet up with other teachers, share experiences, and also support others in your area.

What's to look forward to?

The National Centre will exist until at least 2022, so there's much more to come and to look forward to! The 40 hubs will be established by September 2019.

A new computing resource repository will be available by June 2019, with complete curriculum coverage by July 2020. It will contain schemes of work, lesson plans, activities, and assessment for all key stages, completely free and editable. This will cover all aspects of the ▶

computing curriculum, from KS1 to KS4 (including vocational courses).

For A-Level computer science, there will be comprehensive resources for students and teachers via an online platform for the A-Level programme available from May

will be repeated frequently, so if you miss one, don't worry, it will come round again! These courses are interactive and focused on useful activities that will help you transfer your knowledge into the classroom.

> ## " ONE OF THE AIMS OF THE NATIONAL CENTRE WILL BE TO FOCUS ON HOW WE TEACH, NOT JUST WHAT WE TEACH

2019. Face-to-face events for students and teachers will be starting this year, with the first teacher event being held in Cambridge in January. More events will follow around the country in due course.

More online courses specifically for computing teachers will become available as the months roll on. These

One of the aims of the National Centre will be to focus on how we teach, not just what we teach, and to collate and share the research on computing education in ways that are easily accessible to educators. So do watch out for reports and other information coming out from mid-2019!

How can I get involved?

The team at the National Centre want you to be closely involved with the National Centre. Even if your school isn't one of those running a hub, you can still be involved! Take a look at what's around in terms of face-to-face and online CPD, sign up for CS Accelerator, or go to your local CAS Community. If you want to do more, you can start your own CAS Community and enable teachers in your area to share their practice and learn from each other. Most importantly, sign up at the National Centre website, as then you'll receive regular updates and not miss out on anything! (HW)

Sue is Chief Learning Officer, Raspberry Pi Foundation. An ex-teacher, teacher trainer, and academic, she is passionate about research in computing education.

SCRATCHUP!
A 3D ADVENTURE IN DESIGN AND CODING

Are you looking for new ideas for computing projects? By combining Scratch and SketchUp, pupils can have tremendous fun creating something truly spectacular and visually stunning
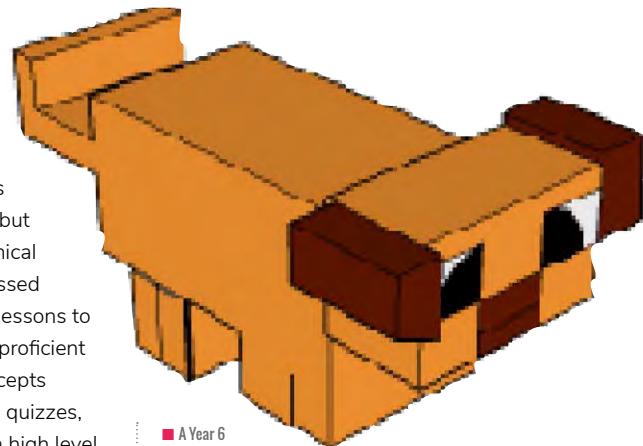
STORY BY Phil Wickins

The idea for 'ScratchUp' emerged after teaching my classes to use SketchUp (**www.sketchup.com** – free software for educators, now owned by Trimble) as a specific computer aided design tool within design technology. So often in primary schools, I've witnessed – with frustration – the CAD objective in the national curriculum being ticked off by teachers asking pupils to create an advert for their product in Publisher or, at best, print out a 2D net of their 3D product. Sadly, this misses the point of CAD entirely, and if we're going to prepare children for industry, we need to get them designing 2D and 3D virtual models of their product before building. I've been teaching SketchUp since 2014 and have always marvelled at how quickly children adapt to it.

In one of my Year 5 classes, however, I was so impressed with their lunar buggy creations that I began to ponder on how they could create something this detailed, accurate, and exciting in one area of computing, while their Scratch projects (**www.scratch.mit.edu** – free block-based programming software) looked like rushed Microsoft Paint attempts. I noticed that their sprite and background
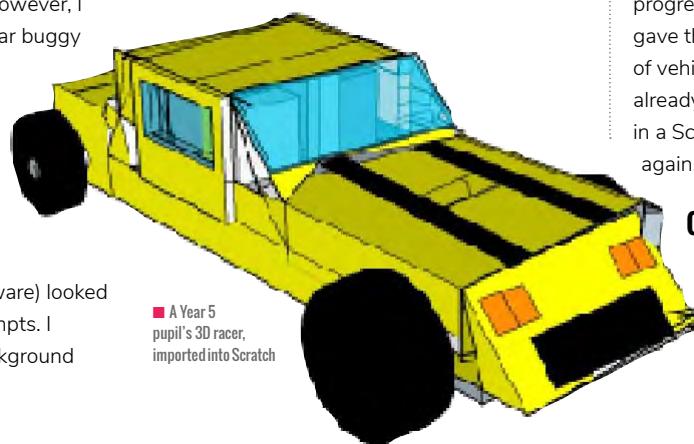
design within Scratch always fell short of their best, probably as they were so keen to code and get their project working. It was all coloured blobs and stick men; purely functional, but little in the way of inspiring graphical user interfaces! They had progressed extremely well in my computing lessons to the point where they were quite proficient using coding constructs and concepts within Scratch. They had created quizzes, animations, and games to quite a high level, so what was the next step?

I decided to capitalise on my pupils' love for 3D design, and use their skills to create high-quality sprites and backgrounds, then export them as 2D graphics to be loaded into Scratch as a costume. As predicted, they spent a lot more time and effort

■ A Year 6 pupil's 3D dog

crafting the detail, adding colours and textures and, most importantly, enjoying the task of creating a sprite. We started off with vehicles (it's slightly easier in SketchUp to create objects with straight edges and flat faces, as well as being a natural progression from their lunar buggies). I gave them free reign in terms of the type of vehicle they would like to create, having already told them it was going to be used in a Scratch project. Their creations were, again, brilliant.

Cheating at 3D - animating costumes
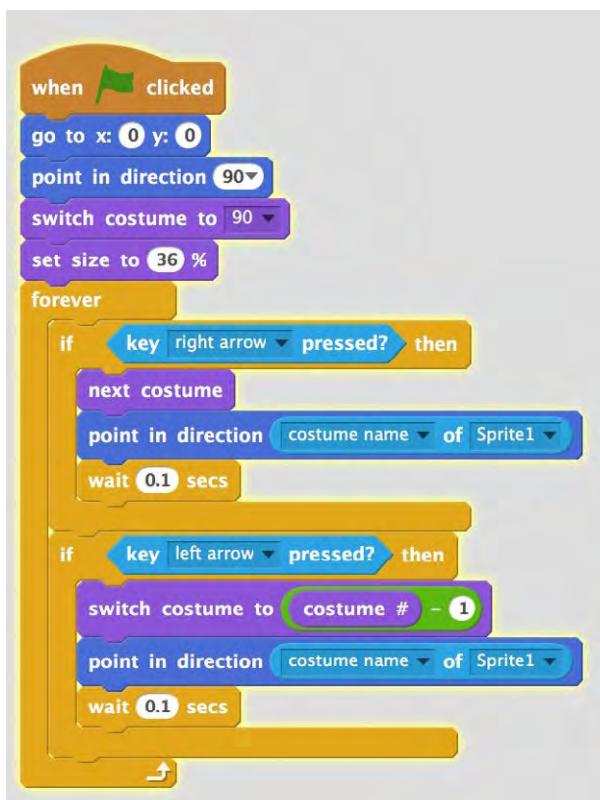
This is the part where the persistence of children pushes you to the point of inspiration.

■ A Year 5 pupil's 3D racer, imported into Scratch

New costume:

1
90
301x99

2
93
290x103

3
99
281x113

4
108
264x123

5
120
228x129

6
140
184x129

7
180
154x123

■ Importing 2D images from SketchUp into Scratch as costumes, then naming each costume with its corresponding angle

■ My prototype 3D car game (See this working at www.rundontwalk.co.uk)

■ Code used to steer the sprite

▶ After having been asked numerous times if they can import their 3D vehicle straight into Scratch (as an actual 3D object) and trying to explain that Scratch only supports 2D images, it occurred to me that we could at least create the illusion of a 3D sprite. By exporting multiple screenshots of our 3D vehicles rotated to a slightly different angle each time, we could 'animate'

our sprites to appear as if they're rotating in 3D space. Having already done some stop frame animation, the children were familiar with this concept and were able to rotate their SketchUp model slightly, export a 2D image, rotate again, export, and so on and then import all of the images, in sequence, into a sprite's costume bank.

I experimented myself first, rotating a simple car that I'd built (very *DangerMouse*, no?). I realised that using the 'next costume' block would be ideal to show the car rotating in one direction. However, to be able to actually control this as a sprite moving around the screen, there would have to be a way of rotating the other way and also moving the sprite in the direction that the vehicle appears to be pointing in. This is where I'm grateful to Scratch, for the ability to give a costume a number AND a name. The costume number enabled me to switch the costumes both clockwise and anticlockwise (using the 'Switch Costume to costume # -1') block combination. However, being able to name the sprite solved the biggest problem of all – how to get the sprite moving in the right direction.

Let's get things moving!

If you think about it, Scratch traditionally has a top down direction setting (up, down, left, right). I always program any vehicle sprites with the same principles you would have if

■ Code used to add perspective - notice you can stop the car by pressing the back arrow

driving a car: up arrow makes it go (accelerator), and left and right arrows turn. So, how do we correspond the direction of movement with the costume? It also occurred to me that even if we can match the costume that looks like the vehicle is traveling directly away from us with the sprite moving up, and the one coming towards us with the sprite moving down, then we have left and right, what about all the angles in between?

This is where the name of the sprite comes in really useful. Using mainly trial and error, I named each sprite with a number. This number corresponded to the angle that the sprite needed to point in, to make it look like it was moving forwards. Then I simply told the sprite to point in the direction of the costume name and move forwards (as you can see from the code on the previous page). Hey presto, we have our 3D sprite moving around the screen!

Finishing touches...
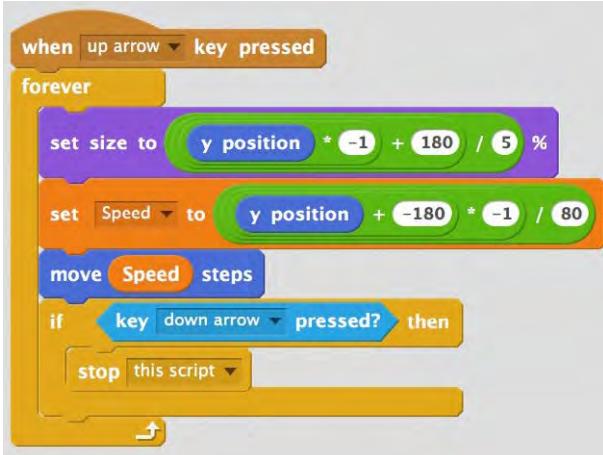
Then to complete the effect, I linked the position of the sprite on the Y-axis to affect both the size and the speed. Meaning, as the vehicle gets further away, its size and speed decrease, to further the illusion of moving in three dimensions. As you can see from the code, it's quite short, yet complex enough for me to turn that into a lesson itself; a lesson that combined maths (3D and coordinates) with art (perspective), and computing. My pupils absolutely loved it and they developed this code together (with a bit of guidance from me), which gave them a massive sense of satisfaction when they began to see their sprites moving around in a 3D environment. They then went on to develop their 3D projects into playable games.

Onwards and upwards!
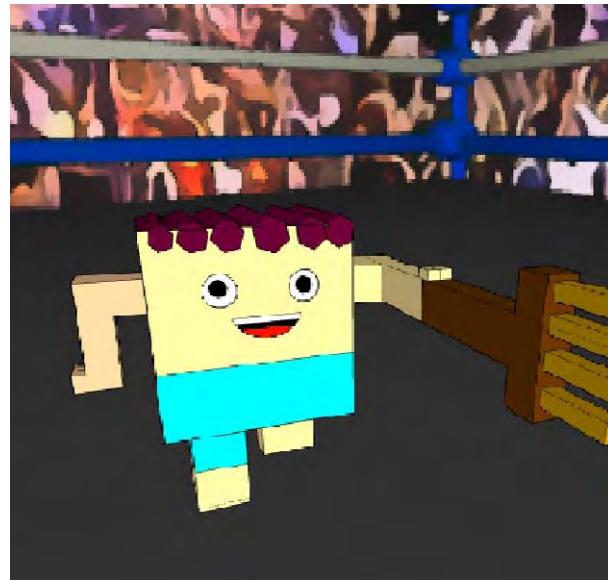
So once your pupils have used Scratch and SketchUp (there are lots of examples of planning and resources online if they haven't), why not try giving them a ScratchUp project? It's by no means limited to vehicles, as you can see – some of my pupils created pixelated dogs and even grannies!

Visit my website **www.rundontwalk.co.uk** for some working examples, and please let me know how you get on; email or use the tag #ScratchUp. (HW)

■ A Year 6 pupil's 3D Granny Wrestler

■ A Year 6 pupil's racer on track

Phil is a CAS Master Teacher based in Southampton, UK. He's been a primary teacher for nine years and now teaches computing as a specialist subject in two primary schools. He delivers Continuing Professional Development for primary school teachers on the computing curriculum and he also has a YouTube channel to support teachers in primary computing, entitled: Delve in, for twelve min!

WHAT'S IN THE BOX?
METAPHORS AND MISCONCEPTIONS

Teaching new concepts is more difficult if the related vocabulary isn't familiar to learners.
Explaining variables using the box metaphor has the risk of misconceptions

STORY BY Jane Waite, Felienne Hermans and Efthimia Aivaloglou

The English KS2 Computing curriculum requires pupils to 'work with variables'. Variables are the first step for students to learn about data structures. Students will encounter many data structures, such as arrays, lists, and binary trees, as they progress into KS3 and beyond. But in primary, we just have to help them 'work with variables'. Simply put, a variable is a reference – pointing to somewhere in memory – where a 'value' is stored.

An example of this is to store a date, say today's date in memory location 1, and our birth date in memory location 2. But to remember which date is where, we use a variable name, such as 'Today's Date' and 'Jane's Birthday'.

The first mental model that pupils develop about a concept is important, as it can be very hard to change and, if it's not right, a mental model can lead to barriers to learning and loss of pupil confidence. Variables are a fundamental idea for programming, so it's worth thinking carefully about what mental models pupils develop for this concept.

Variable is a term that students might come across (this is called topical word learning) in a range of contexts, such as in colloquial use as something that 'varies', or in other school subjects, such as in science or maths. However, each of these contexts has subtly different meanings to the computer programming concept of a variable.

All figures by Jane Waite and box from Pixabay CC0 Public Domain

■ Figure 1: Create a variable - make the box and label the box

One way to explain computer programming variables is to use a metaphor. Metaphors often have a 'shelf life' – they work for so long and then they need to be replaced, as there will be differences between the working of the metaphor and the concept we are linking it to. This is true of the metaphors and methods used to explain programming variables.

A common metaphor used to explain variables is the box metaphor. This seems like an attractive way to make a new word more familiar.

How does the metaphor work?

The box metaphor goes as follows:
1. As shown in Figure 1, the box is made and a label is placed or written on the box. The box is the variable, with a variable name – we've created a variable
2. As shown in Figure 2, values can be placed in the box – simply put, these might be numbers, text or yes/no type values, and can be the initial value or updated values
3. As shown in Figure 3, values can be retrieved from the box by looking inside.

Figure 2: Set the values, store values, change values, update values - put the values in the box

So why might things go wrong?

Firstly, pupils might think that more than one value can be in the box at any point in time. For example, if 1 is added to a score, they might think that the original value and the 1 are now in the box. They might also think that to find out the current value of the variable, they need to perform some kind of action to add up all the numbers in the box. Similarly, if a text value is put in a box, then pupils might think that the value that was there before is still in the box, and there are lots of text values. This might lead them to create a mental model where a variable has not just one value, but lots of values and they can access the old values too.

A variable, however, doesn't hold multiple values; a variable 'references one place in memory' and 'holds' just one single value at any point in time. The old values that were there before can't be retrieved. Simply put, as a variable is written to, the old value is overwritten.

What did the research find?

At a large museum in Amsterdam, the research team set up an experiment. They taught nearly 500 participants, two-thirds of whom were children and a third parents, about variables using Scratch. Half of the participants were taught using the box metaphor and half using a label metaphor. All participants were then asked some questions. One question specifically targeted whether

" IF A TEXT VALUE IS PUT IN A BOX, PUPILS MIGHT THINK THAT THE VALUE THAT WAS THERE BEFORE IS STILL IN THE BOX

they thought a variable could hold more than one value. Those who were taught using the box metaphor were far more likely to have the misconception that variables hold multiple values for text ▶

Figure 3: Select, retrieve, get values - look in the box

COMMON MISCONCEPTIONS

The misconception of 'thinking that more than one thing is in a variable' can be particularly re-enforced if an unplugged activity is used to explain the metaphor. For example, if bean bags are thrown in the box to represent adding individual points to a score, or if pieces of paper are thrown in a box to represent the next answer to a question, then learners can see and may even have physically enacted this multiple values in a variable idea.

A second misconception is that when a value is retrieved from a variable, then the value is no longer in the variable - it's no longer in the box.

It has been taken out. As you can see in Figure 3, this misconception is attempted to be reduced by showing a telescope to represent retrieving the value and the phrase 'look in the box' is used. But it's easy for children to think they can actually take the value out of the box, particularly if an unplugged activity involves taking bean bags or pieces of paper out of a box to see what the value is.

A variable doesn't lose its value when the value is retrieved - they continue to hold the value, whether the value is retrieved or not. The value persists until the value is overwritten by a new value, or the value is deleted.

---

values. However, for a simpler question on what value had been first stored in the variable, the box metaphor participants performed better than the label metaphor group.

It seems that the shelf life of the box metaphor was reached as soon as more than one value had been saved to the variable, but to start with the box metaphor was a better way to get the basic concept of saving the first value into a variable. If you want to find out more about the research, led by Felienne Hermans and Efthimia Aivaloglou at Delft University and The Open University in the Netherlands, read about it in Felienne's blog **helloworld.cc/2PS47Mw** which includes a link to the research paper.

What other metaphors or methods are there to explain variables?

A variable is a label. Here the variable is explained as being a label, like a temperature or the name of a person. In the Dutch research on variables, they explained this by always showing 'x is 5' compared to 'x contains 5' for the box metaphor.

Hoops Similar to the box, but rather than using a box, a large hoop is used. This means you can more easily see the values 'in the variable' and you can throw things in it, like bean bags representing points for a score. Can you see what misconception this might lead to?

Thin tubes Rather than a big box, a thin tube is used, with cards used to represent the variable, where the cards snuggly fit in the tube, so that only the last variable put in the box can be seen. This still implies there are lots of values in the variable, and perhaps worse still that there's an ordered stack of previous values in the tube.

Paper, pegs and a washing line Write the value on a piece of paper, write or fix the variable name on the peg. Peg the value

to the washing line when a value is set, unpeg when a value changes, and peg up the new value.

Mini whiteboard Write the variable name in small letters at the top of the card. Write the value in big letters on the board. When a new value is set, wipe out old values and write the new value to replace it.

There are other metaphors and methods – what method do you use? Maybe share your ideas on Felienne's blog – she would love to hear about your approaches.

Which is the best?

As far as I know, there has been no research to compare all the metaphors and methods available to teach variables (but Felienne's team are looking at this now). Nor has there been research to look at the long-term impact of the different approaches for different programming language, nor of the long-term impact on pupil confidence of using metaphors which have a 'short shelf life'.

I personally like the whiteboard method, as primary pupils are familiar with using whiteboards. Whiteboards are often available in primary classes, and using whiteboards to keep scores is perhaps a common idea in games and quizzes. Using quizzes to introduce variables is a common context, so this method is easy to link from unplugged method to programming activity.

Using metaphors and unplugged methods for helping children learn concepts seems like a good idea, but there can be misconceptions lurking. Misconceptions may be acceptable for a while – we may decide that being able to start the learning process off from an easy start point is worth the problem of having to address the misconception later. However, as teachers, we need to be aware of this, so that we plan in the 'unlearning and reteaching', and

> ## USING METAPHORS AND UNPLUGGED METHODS SEEMS LIKE A GOOD IDEA, BUT THERE CAN BE MISCONCEPTIONS LURKING

be aware of potential barriers to learning which we may have put in place because of the mental models that we have helped learners develop.

If you would like to find out more about metaphors and misconceptions, you can read about them in the Barefoot variable concept document. Try the Introduction to Variables Unplugged activity (it uses the whiteboard method), or sign up for one of the primary CPD sessions with the National Centre for Computing Education (NCCE). Or ask at your local NCCE hub, CAS local community, or talk to your local CAS Master Teachers. (HW)

RASPBERRY PI-MONITORED
CHEMICAL REACTOR
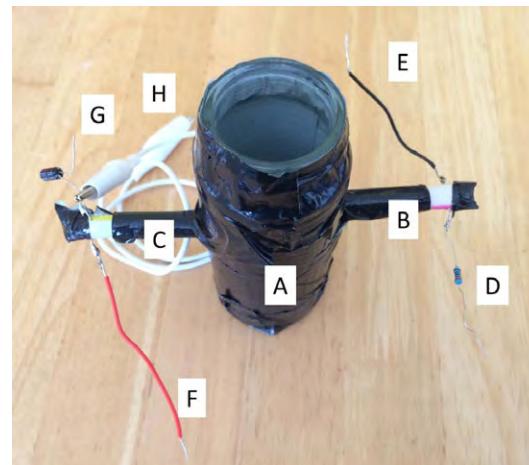
A Raspberry Pi can be used to monitor the reaction between hydrochloric acid and sodium thiosulphate to complement a popular GCSE chemistry practical

STORY BY Steven Weir

T he rate of reaction between hydrochloric acid and sodium thiosulphate is typically studied as part of GCSE chemistry. The experiment involves measuring the time required for the reaction mixture to turn cloudy, due to the formation of sulphur as a precipitate. Students can then change the temperature or concentration of the reactants to study their effect on the rate of reaction. The time for the reaction mixture to turn cloudy is normally facilitated by recording the time a hand-drawn cross takes to become obscured when placed underneath a glass vessel holding the reaction mixture. This timing is prone to variability due to operator judgement of when the cross first becomes obscured. This variability can legitimately be discussed as part of the lesson. However, the element of operator judgement can be avoided using a Raspberry Pi-monitored chemical reactor.

The chemical reactor

Attached to a glass jar of approximate 80ml volume (the size is not critical) are two drinking straws, of which one houses a white LED (light emitting diode) and the other a

■ Figure 1: The chemical reactor:
A: Reactor covered in black tape
B: Drinking straw attached to the reactor, with a further straw inserted housing a white LED
C: Drinking straw attached to the reactor, with a further straw inserted housing a LDR
D: 220Ω resistor to connect to the LED and GPIO 23
E: Wire to connect to ground
F: Wire to connect to 3.3v supply
G: 1µF capacitor to connect to ground
H: Crocodile clip to connect to GPIO 27
(NB the other end of the wire is situated in between the capacitor and the LDR)

LDR (light dependent resistor). The jar is covered in black tape to minimise intrusion of ambient light. The reactor is shown in Figure 1, along with details of other electrical components and connection instructions to a Raspberry Pi.

Results

The Python code shown in Figure 2 should be run prior to addition of chemicals to the reactor. Instructions appear on the screen to prompt chemical additions and to start data collection.

Figure 3 shows the results from the experiment when 25ml 0.1M hydrochloric acid is reacted with 25ml 0.15M sodium thiosulphate at 20°C. The reaction is complete at the time the light transmission first reads 0, (i.e. complete obscuration of the light by the precipitate formation) – in this example, that time is 45.4s. For more able students, tangents can be drawn at various points on the curve, and gradients calculated to determine the maximum rate of reaction from various reaction conditions. **(HW)**

Steven is a Science Tutor working in Halifax and Huddersfield.

■ Figure 2: Python code for the chemical reactor
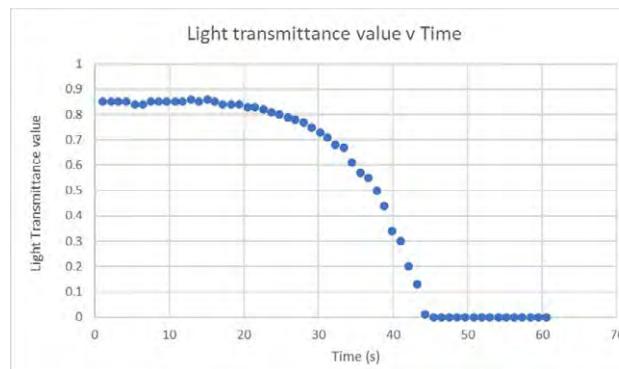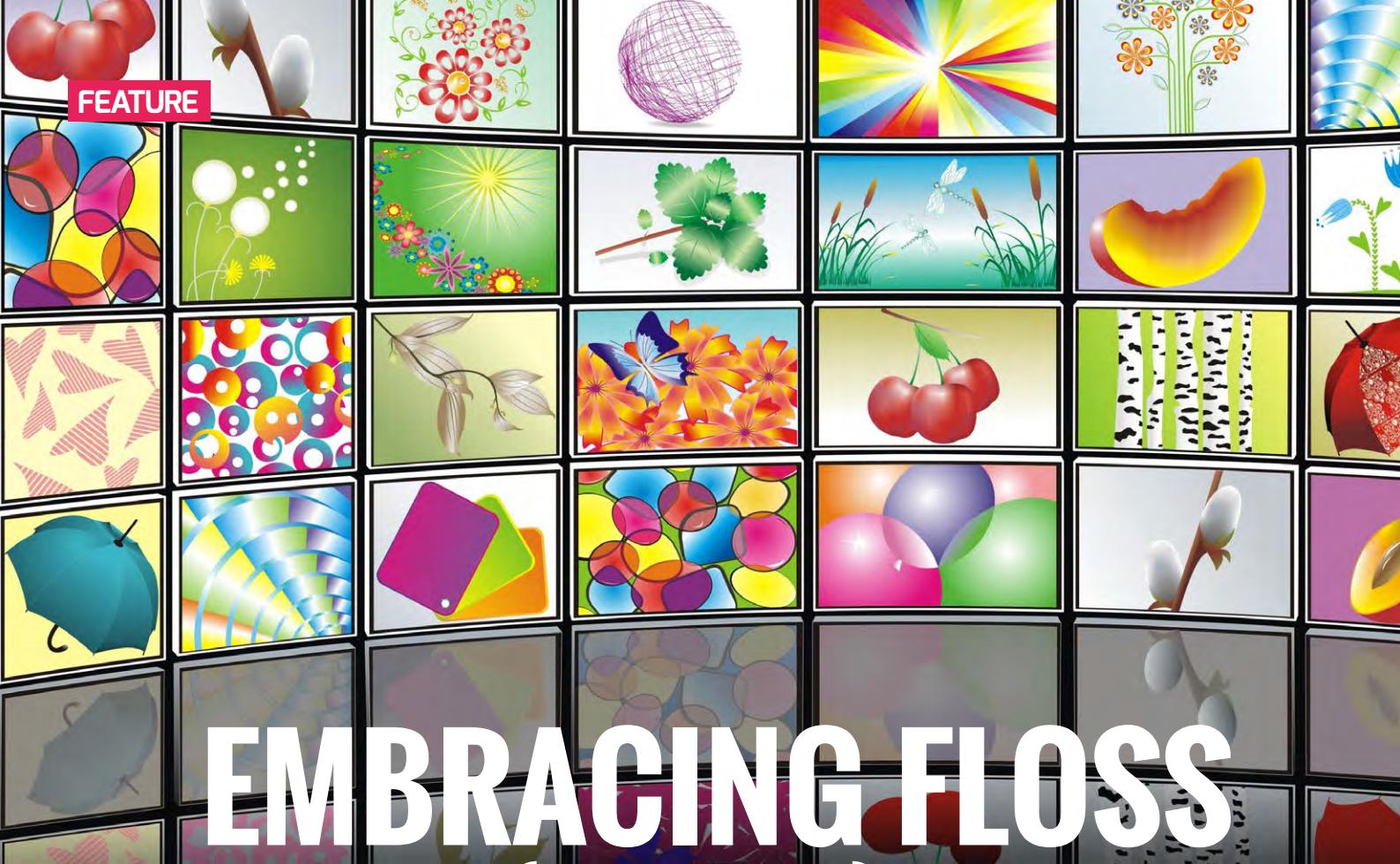
■ Figure 3: Graph showing the change in light transmission with time

EMBRACING FLOSS
FREE (LIBRE) OPEN SOURCE SOFTWARE

I'm increasingly using free software. In an age of free(mium) web apps, I seem to be going against the flow

STORY BY Paul Powell

A few years ago, I was using a bit of free software along with a broad selection of proprietary applications. I had a couple of issues:

1) The proprietary apps wouldn't work on Windows 10 without an upgrade, or were no longer developed

2) Some free web apps were changed, restricted, or moved to a subscription model

I had nobody but myself to blame. Nobody promised me that it would stay the same or it would always be free. Except, of course, for free software.

Student access

I had used some open source software before, but only because I was too miserly, or it was well known. Perhaps it was time to move more in this direction. Students accessing tools at home is important. If I teach video editing in my lesson, they'll learn a little. If they decide to use those skills to start a YouTube channel, they'll learn a lot more.

Some software we used was too expensive for students to invest in and not all of it was available on Mac. Free software usually was.

Some students don't have a computer, but most the software we use now can run on a Raspberry Pi. Suddenly, something that was meant to get around inconveniences started looking like it could help address the digital divide.

Budget

One major area of budget expense is software. I imagine most of us haven't

QUICK WINS

Some swaps are easy and are worth giving a try. Here are a few:

- **Stop-motion animation** Pencil 2D is a quick cut-down animation package. It supports onion skinning, and is good for quick and easy editing.

- **Key-frame animation** Synfig is vector based and supports tweening and bone tools. There are some impressive demos.

- **Vector graphics** Inkscape has been around for years and is a fantastic vector editor. Tracing a photo into a vector to create an avatar is always a popular lesson.

- **Image editing** GIMP goes from strength to strength for image editing. Version 2.10 finally goes for a single window interface. Highest on the roadmap for the future is more support for non-destructive editing, closing the feature gap significantly for most users.

- **Video editing** Kdenlive has been ported to Windows and is in beta. We're using it, and it's proving capable and relatively intuitive.

- **Voxel game** Minetest is an open source clone of a similarly named game. Various add-ins (also open source) bring items like logic gates into the game world.

Courtesy of The GIMP Team – Licence: cc-sa-4.0

■ Modern GIMP is slick and effective. Worth another go if you've only tried an older version

seen our budgets increase. I've managed to reduce my software spend to zero and yet now we have a richer curriculum. The open source tools get better with each release, so now I'm actively seeking to replace every tool with an open source one. Our OS is set by the school, but I'd like to be able to run the whole curriculum on a computer that costs less than £40.

> ## " I'VE MANAGED TO REDUCE MY SOFTWARE SPEND TO ZERO AND YET NOW WE HAVE A RICHER CURRICULUM

Belief

Free software is distinguished by granting users the right to read, modify, and distribute the software in question. But all of the advantages I've mentioned are about flexibility and price, rather than anything to do with the freedom, right? Actually, I think

these advantages all flow from the freedom granted by the authors. Cross-platform support is common because anyone can have a go at building it. Changing how the product fundamentally works usually results in a spin-off project to maintain a version closer to the original. Suddenly charging for the product

won't work because someone else has the right to distribute it for free.

With free software becoming increasingly able to replace proprietary software for most of us, perhaps we could push it a bit more. I've even started tinkering with making some modifications. Unfortunately, the one thing that isn't free is my time! (HW)

Paul is Curriculum Lead for Computing at George Mitchell School in East London. He is becoming increasingly evangelical about FLOSS.

PROGRAMMING PEDAGOGY

THE BEST WAY TO TEACH PROGRAMMING?

Alan O'Donohue has a collection of ideas for teaching
programming to a mixed ability class of students...

"**T**he role of the teacher is to create the conditions for invention rather than provide ready-made knowledge" - Seymour Papert.

Why should I read this guide? For those learning how to program a computer, there are many different learning routes available, but teaching programming to a class of 30 mixed ability students is a very different matter from learning how to program. This article compares a variety of strategies for teaching programming with varying degrees of success. The guide should help even the least confident among computing teachers select some strategies to support them to teach

programming to their classes. The guidance within this article is aimed broadly at teachers of Key Stage 2, Key Stage 3, and Key Stage 4. Bear in mind that this guide represents the opinions and experiences of the author and is neither exhaustive nor prescriptive.

What if I'm too busy to read the full guide? In short, to develop successful problem-solvers (and programmers), teachers should plan lots of repetitive practice through a variety of investigation activities and problem-solving exercises for their classes, building in frequent opportunities for students to reflect on the successes of their solutions.

What does the word 'programming' refer to? Within the context of this article, I use the word 'programming' to refer to many more activities than just the activity of text-based programming itself. In computing education, the word 'programming' is frequently used as an umbrella term to describe a group of problem-solving activities that include computational thinking, algorithmic design, and developing a coded solution to an identified problem.

How important is it for computing teachers to be able to program? In my work supporting schools, I've noticed a tendency for some teachers who lack previous experience of programming to focus, perhaps disproportionately, on the coded element of programming. This may be because some teachers have identified this as an area where they currently lack expertise, and this induces a sense of fear or panic. To illustrate this, I was surprised when I encountered a non-specialist Key Stage 1 teacher enrolled on a 'Teach Computing with Python' course I was leading a couple of years ago. She had been informed by a senior leader that she should be teaching Python to her Year 1 class in order for them to succeed in computing and to really stretch the class!

There's a danger that placing too much emphasis on teaching students how to develop coded solutions may mean neglecting

to teach computational thinking and algorithms. Computational thinking includes a range of critical problem-solving skills that can be applied to many situations, not just those requiring a computer. In fact, many aspects of programming and algorithmic design can be taught in a manner that requires neither computers nor programming languages. For further reading, refer to CSunplugged.org (**www.csunplugged.org/en**).

There are many stages involved in developing an effective coded solution, and many good habits and strategies that teachers should encourage their students to follow, for example identifying the essential and desirable requirements of the problem in the first place. If a student isn't clear what problem they're trying to solve, the solution they develop may not actually solve it. Equally, the ability to develop an effective testing strategy is another important skill – this helps students to evaluate how well their solution performs when presented with data other than that which is expected in order to test the robustness of the developed solution.

So, what is the best way to teach programming? There are many best ways to teach programming, since there are a broad range of different approaches and teaching strategies available, and while this guide isn't exhaustive, it highlights some of the more popular approaches that computing teachers are using. There's no single approach that will suit every teacher in every setting. There's a great deal of overlap between some of the strategies described below. Teachers are more likely to find over time that different approaches suit different settings, scenarios, and the varying needs of different groups of learners, as well as their own comfort and confidence levels. As part of their own professional development, teachers should feel encouraged to experiment with different approaches in their own teaching until they find a flexible strategy they feel comfortable with.
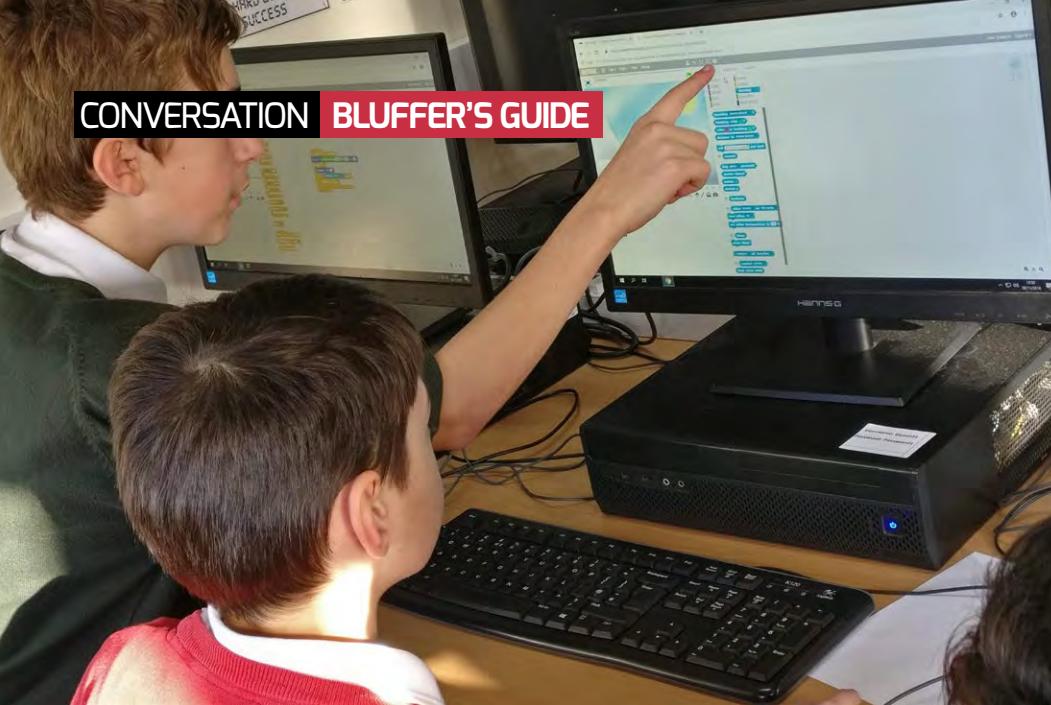
What difficulties are teachers likely to encounter? It's no easy feat teaching students effective problem-solving skills; it requires a large commitment of preparation, planning, effort and energy, and lots of repetitive practice with students over a long period of time. However, some teachers may find themselves in very challenging circumstances, where there are unrealistically high expectations for

results, the pressures of reduced time, and an atmosphere of high accountability. In these stressful situations, there's an overwhelming temptation to simply teach students the answers to the problems or provide them with pre-prepared solutions, rather than teach the methods that students can use to get there under their own steam. While this option may appear to solve immediate issues, it will almost certainly lead to longer term, potentially serious consequences. In one recent example, schools witnessed an increase in malpractice investigations into the teaching of GCSE Computer Science when it appeared that many students had developed near-identical solutions to the same set of problems set by the exam board. While it's inevitable that students working on the same problem are bound to develop similar solutions, in some cases these students were incapable of explaining how they themselves had arrived at their solution and presented solutions they themselves didn't understand.

Additional challenges encountered teaching programming:
- **Few classes are the same as any other** – since students hail from different backgrounds and their prior experiences vary, this can have a strong effect on their levels of confidence and perceived abilities. To remedy this, build student confidence by planning to set problems broken down into smaller more manageable chunks, ensuring that the first few problems are accessible to all students.
- **Successful problem solving requires students to demonstrate persistence, resourcefulness, and initiative** – unfortunately, these qualities don't come instinctively to all students. When I've encountered this issue with some of my classes, I've put the computers away and planned whole-class problem-solving activities to highlight these qualities and to develop these characteristics within my students.
- **Lack of teacher experience** – even teachers who've been teaching computing for five years or more may only have been teaching programming for a few years, while our colleagues in other subject areas have access to a wealth of knowledge and experience of teaching their subject. Thankfully, there have been recent increases in the amount of studies being published into researching which are the most effective programming pedagogies. You can keep on top of the latest studies by reading issues of *Hello World*. ▶

What strategies are there to teach programming? What follows below is a list of popular approaches used in different settings and scenarios; not all of them are presented as recommended strategies. Also, don't view this list simply as discrete strategies, since some of them can be blended or combined to achieve different effects. If there's one you haven't tried yet, you might decide to explore it further and share your conclusions with other teachers in your team or at a CAS Meeting:

1. **Using prepared resources and worksheets** – When students work through activities following worksheets provided by their teacher or online coding instructions on a website that teaches "how to code", often the instructions have been created by someone other than their teacher, for example "Type this in... now type this to make x happen".

■ **Advantages** – There's an abundance of materials already available, reducing the amount of advance preparation required. These activities don't require a lot of support, input, or expertise from the teacher, and they don't rely heavily on teacher knowledge or experience either. When pupils encounter a situation in which their code doesn't work, it may be easier to spot the error as the resource has already demonstrated the perfect code solution. As such, if the student's doesn't work, it's because they've made a mistake copying it down. Often adopted in scenarios when teacher expertise is quite low, such as Coding Clubs led by volunteers.

■ **Disadvantages** – This approach isn't particularly exciting or stimulating for students, and there's a heavier emphasis on completion of activity, rather than deep learning taking place. It's unclear how much information and understanding the children actually retain afterwards. The approach leads to students becoming heavily dependent on the provided resources and means there's often little difference between individual student work. This can also lead to problems related to assessment; when all students have effectively the same work, how can you discern which students have actually learned the most from it? In this scenario, teacher assessments may be biased toward the only one perfect solution, and it has the unintended effect of suggesting to learners that there's only ever one solution to the problem. We know very well that there are often many different solutions to the same problem, each with its own merits. Students may complain that they've only learned how to do one discrete activity, not apply what they've learned to another context.

2. **Follow the teacher** – The teacher usually starts typing in a particular piece of code and then demonstrates what it does to the class. Then the teacher explains what they're doing and asks their students to repeat the exact same exercise on their computer and test it.

■ **Advantages** – Doesn't require that the teacher does a lot of future planning or preparation of resources, as the teacher can pretty much decide what the learning is shortly before the lesson. When the teacher makes mistakes, it helps the students to see that errors are an everyday part of programming.

■ **Disadvantages** – This approach offers very little intrigue or surprise. The joy of discovery is robbed from the learners; if they've already seen what happens, there will be little excitement or interest value, unless of course it doesn't work when they try it themselves! Doesn't easily allow for students to progress at their own individual pace, since the students are reliant on the teacher to show them the next steps.

3. **PRIMM** – See article in *Hello World* issue 04, PRIMM - Predict Run Investigate Modify Make. This is best viewed as a set of nested activities rather than one single strategy, meaning that the teacher doesn't necessarily need to follow all five steps each time in a linear fashion or even in the same order. So, a teacher might plan for a series of investigation activities first, then ask the students to make some modifications, predict the outcomes of these modifications, and then run to see how accurate their predictions were.

■ **Advantages** – The learning impact is backed up by studies as well as anecdotal evidence, which agree that this is an effective approach to teaching and learning programming.

■ **Disadvantages** – None apparent, as long as the teacher doesn't slavishly prescribe all five steps be followed in strict order.

4. **Step-by-step or build-up exercises** – The teacher may provide their classes with a booklet of exercises, each one requiring a solution before moving on to the next, for example "First complete exercise 1, then move on to exercise 2". This approach aligns with the concept of 'Assessment 4 Learning', and helps students recognise what they can already achieve and at the same time helps to highlight apparent gaps in their knowledge, understanding, and skills.

- **Advantages** – Linear process makes it easier in some ways to track, assess, and monitor learning progress and achievement. It enables students to take more responsibility for their learning, since they can see their own progress as well as future obstacles to their own learning. Allows students to progress at their own individual pace.
- **Disadvantages** – If a suitable booklet doesn't already exist, then it requires the teacher to spend a lot of time creating the resource.

5. **Challenge based** – The teacher sets a number of stepped challenges for the class, which increase in difficulty and challenge level. All the class may be working on the same theme or context, but working on different levels of complexity of the problem at the same time. The challenges may outline broad problem scenarios, with opportunity for students to solve parts of the scenario in stages.

- **Advantages** – A range of possible outcomes enables students of all levels to achieve an element of success. The different solutions that students ultimately arrive at provide a valuable resource for critical appraisal.
- **Disadvantages** – Requires a lot of teacher preparation to ensure that the challenges match and stretch the abilities of the class.

6. **Modifying existing programs** – Similar to the PRIMM approach described above, the teacher starts by directing the class to examples of programs that already exist, often these may be games. Then the teacher asks the class to predict the effects or what may happen if certain parts of the program are modified. The class then test their theories. The teacher may then ask the class to try to solve other challenges by modifying other parts of the existing programs to achieve varying outputs.

- **Advantages** – When using existing programs, there isn't a lot of resource creation required. The students can find this approach very stimulating, especially since they haven't had to start with a blank screen and create their own program. When errors creep in, the students can refer (or revert) back to the original provided program. Students are generally quite engaged in this approach as there's a fun element in changing and modifying existing code, particularly as it doesn't rely on the student understanding all of it.
- **Disadvantages** – It requires a lot more thinking and planning ahead if this approach is being used regularly to support a list of intended learning outcomes in a heavily structured manner.

7. **Paired programming –** See the article in *Hello World*, issue 04. May be used in combination with some of the approaches listed above. Students are placed in matched pairs to arrive at a solution together. Each student is assigned the role of driver or navigator and they take on each role in turns at timed intervals while working towards common goals.

- **Advantages** – Over a period of time, it takes a lot of the pressure off the teachers, since students naturally support each other, and build up their confidence and problem-solving abilities in pairs. Students are less prone to make errors as there's more than one pair of eyes reading the code on screen. A lot of problem solving takes place 'off-screen' in the discussions between the pairs.
- **Disadvantages** – At the beginning, it requires a lot of explanation and enforcement until the students become accustomed to working in pairs and other good habits. Intended as a way to support student learning and progress, some teachers haven't easily identified ways to assess the student learning progress, but this is resolved if the paired activity is viewed as learning rather than an assessment opportunity. **(HW)**

FURTHER READING

PRIMM - *Hello World*, issue 04
Pair Programming - *Hello World*, issue 04

■ The Computer Literacy Project led to the introduction of the BBC Micro

CELEBRATING HISTORY

The introduction of the computing curriculum is a welcome step forward,
but have we missed an opportunity to celebrate our history?

STORY BY Gary McNab

The introduction of the computing curriculum saw a sea-change in attitudes towards the subject area. While British business raised concerns that children were leaving school with office-based skills that didn't meet the required skill set of the labour market, those charged with delivering the subject to children saw the ICT curriculum replaced by one designed to support children in an ever-changing world of work with the introduction of computer science.

But is this a case of history repeating itself? Some readers will remember a similar situation in the late 1970s, when the country approached a fresh decade with the advent of new working practices, skill sets, and a competitive edge required by business, together with the introduction of the microcomputer to the world of industry and manufacturing. It brought with it a fear of change within businesses to adopt

this new technology, and the requirement to upskill the population in readiness for the new technological age. But behind the turmoil of an ever-changing workplace, both historically and in today's digital market workplace, have any of us stopped to think how did we get here?

Innovators

Charles Babbage, Ada Lovelace, Alan Turing, and Tommy Flowers – all names accredited with the dawn of logical thinking, programming, and debugging, and all rightfully embraced and celebrated for their achievements.

The United Kingdom is a pioneering nation of inventors and innovators, and while the history curriculum has a requirement to teach about the Egyptians and Romans, for example, given our achievements, should the computing curriculum not embrace our history of computing prowess?

At the end of the Second World War, a small-scale experimental machine nicknamed 'Baby' was created by the University of Manchester. Using surplus hardware from the war, a functioning prototype was created that eventually led to the Ferranti Mk1, the world's first commercially available computer, and with it the world's first program stored within its memory.

If we fast forward to the 1950s, we have the world's first business computer – the Lyons electronic office, or 'Leo'. It was designed to calculate the cost of bakery distribution within the Lyons empire. So the computing revolution started within the tearooms of Great Britain! Such was the demand for tea and biscuits, there was the realisation that automation was the only way forward. Indeed, the computer was eventually programmed with punch cards to do the company's payroll.

Circuits

The 1970s brought real change in technological advances with integrated circuit and silicon-enabled printed circuit boards, along with CPUs that eventually brought the dawn of affordable computing to the home. The Nascom 1, a British computer introduced in the late 1970s, was ready and waiting for you once you had soldered over 3000 parts to the circuit board!

The radio hobbyists of the time were the ones who adopted this new technology initially, but a certain Sir Clive Sinclair at Sinclair Instrument Ltd (later to become Sinclair Research) saw the potential for an affordable programmable computer. Sinclair subsequently designed the ZX80, and launched the machine in January 1980. The birth of the information age and the forthcoming home computer boom would take the entire country by storm. Not long after, the BBC launched the Computer Literacy Project, a programme designed to upskill the nation, and it awarded a contract to Acorn to design the BBC Micro, which would become the staple diet of computing in 1980s classrooms. That followed a government initiative to promote the use of computers in school, allowing children to leave school computer literate.

While the BBC Micro would eventually lose out to the IBM PC, the effect British inventors had on 1980s children and teenagers can't be underestimated. From the Sinclair range of ZX80, 81, and ZX Spectrum computers, to the Welsh Dragon 32 machine built in Port Talbot (seen as the
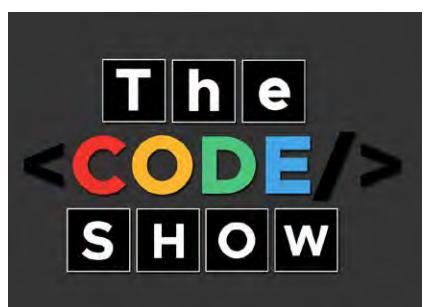
■ Sir Clive Sinclair saw the potential for an affordable programmable computer

Ocean. They were helped by the demand for videogames (after their homework was completed!). After all, these UK companies helped to create what we now know as the videogame industry. The very same industry that contributed £1.5 billion to the national GDP in 2017.

So why is this relevant? Well, given this very brief historical timeline of British

the technology was and how far it has come in such a short period of time.

There's a saying: "from small Acorns grow mighty trees". To conclude in this case, from small Acorn computers, mighty ARM CPUs power Raspberry PI computers! (HW)

> **GIVEN OUR ACHIEVEMENTS, SHOULD THE COMPUTING CURRICULUM NOT EMBRACE OUR HISTORY OF COMPUTING PROWESS?**

Welsh Silicon Valley of its time), through to the Oric 1 and Atmos computers, along with Camputers Lynx, Elan Enterprise, Sir Alan Sugar's Amstrad CPC range, the home computing boom created bedroom coders. Coders who were often self-taught, who created their own software, which would ultimately create software houses such as Gremlin Graphics, Imagine, and

history, we can explain to our current generation of learners that what we're teaching in computing isn't a new concept, it's something as a nation we're historically very good at. While not everyone will become a coder, we can help inspire graphic artists and musicians, for example, by giving children hands-on access to this hardware. That way, they can experience how limited

Gary is founder of The CODE Show, a travelling computing museum that visits schools to give children hands-on access to historical computers, which tells the story of the home computing boom.
www.thecodeshow.info
email: hello@thecodeshow.info

COLLABORATION AND COMMUNICATION IN THE COMPUTING CURRICULUM

We'll examine ways to provide opportunities for talk, and to enable pupils to collaborate effectively in lessons

STORY BY Neil Rickus

T he English computing curriculum states pupils must be able to "produce content" and "express themselves" (DfE, 2013), which can be used to facilitate a range of collaboration and communication opportunities. In particular, having technology available enables pupils to develop a variety of social skills, such as communication, negotiation, problem solving, and collaboration (Reid et al, 2002). To engage pupils in computing lessons and ensure coverage across the curriculum, there's a need to undertake creative projects in a "fun and collaborative environment" (CAS, 2017) and, when tasks undertaken are relevant to the pupils' interests, they can undertake "exploratory talk" (Mercer, 2008 [1]) within the safe and secure environment of the classroom. Therefore, how can we provide opportunities for collaboration when teaching computing?

Computational thinking and program development

Barefoot computing (CAS Barefoot, 2014) defines computational thinking, which underpins the computing curriculum, as a combination of six concepts and five approaches, with each area providing opportunities for pupils to work together. Barefoot's 'collaboration' approach could involve a range of pedagogical techniques, such as paired programming, which allows pupils to develop their programs together, with one pupil solely using the input devices, while the other pupil focuses on the required instructions. In addition to this, collaboration complements a number of other computational thinking approaches, such as 'debugging' and 'persevering', which may not be as effective if approached independently. Teachers have also noted the "positive motivational impact" (Sentance et al, 2015) collaborating on programming tasks has on individuals and the class as a whole.

In DT, the "Engineering design process" (Bers et al, 2010) is often followed, with

■ Pupils discuss bugs, or errors with their program's code, while working together at the computer

the stages of 'Imagine' and 'Plan' allowing children to articulate their ideas and share their thoughts with others. Such a process is increasingly being used when implementing physical computing projects, which ensures pupils have carefully thought about both the physical and programming elements of their project before beginning work. The process also allows children to regularly evaluate and improve their work, which lets them "talk [the outcome] into existence" (Dawes et al, 2006, p113) through the various iterations of their project.

Teacher input

When teaching programming, teachers need to ensure they focus on the concepts being taught, rather than the hardware or software been used, as pupils can become disengaged when using the same technology on an ongoing basis (Bagge, 2013). Through the use of technical language, pupils' familiarity with terms is increased, which is essential as they produce more complex projects, and move to using text-based programming environments (Kölling et al, 2015). This can also allow pupils to clearly articulate how their programs function when working with others, and to facilitate the debugging of programs.

During paired programming and other collaborative teaching approaches, such as 'C3B4ME', where pupils have to ask three peers for assistance before the teacher, teacher modelling is needed to demonstrate to pupils how they should interact (Bird et al, 2014). This enables children to ensure they empathise, listen, and potentially continue their conversations in more depth, rather than the exchange ending in a dispute (Littleton, 2013). Teachers with limited experience of delivering computing lessons can use these pedagogical techniques to act more as a facilitator, rather than the knowledge bearer, and can help to avoid them being inundated with "lazy questions" (Bird et al, 2014, p69). However, this process needs to be carefully managed to ensure more able pupils aren't continuously disturbed during lessons, and that the lesson is challenging for all pupils.

Block-based programming environments

TALK DURING COMPUTING

Speaking and listening activities need to be carefully managed during computing lessons. Giving the children too much freedom can lead to off task behaviour, while limiting collaboration reduces opportunities for pupils to learn from each other. We work carefully with pupils to model the appropriate language for talk, and encourage them to use technical vocabulary where possible.

can facilitate further speaking and listening activities. For example, most environments allow sound to be recorded for inclusion within pupils' programs, and a spoken narrative can often be recorded while the program is on the screen.

Introducing IT elements

Computing unplugged activities enable pupils to develop their understanding of computational thinking without the need for technology. Many unplugged activities, such as the "Sandwich Robot" (Bagge, 2012), require pupils to limit their vocabulary choices and give precise, unambiguous instructions, while developing their understanding of the computational thinking areas of 'abstraction' (removing unnecessary detail), 'evaluating', and 'collaborating'. These activities also provide opportunities to introduce other technologies into lessons, such as the camera on a tablet device, for pupils to film their peers in role to assess the quality of their instructions and aid debugging.

The use of IT to record audio and video can also enhance other areas of the computing programme of study. E-books containing multimedia content created by the pupils provide opportunities to share their work with an audience beyond the classroom and, when considered in conjunction with Papert's theory of Constructionism (Papert, 1993), this use

of technology is likely to enhance pupil outcomes by providing an audience for their work. When pupils are including information gained from their own online research, the most successful pupils use "exploratory talk" (Knight et al, 2015, p303) to assist with "sorting out his or her own thoughts" (Mercer, 2008 [2], p18).

Other recording technologies, such as "talking picture" apps (Byrne, 2017), which allow recordings to be synchronised with the movement of a character's lips, can enable pupils to demonstrate their understanding of curriculum areas without the barrier of having to produce written content. For certain pupils with Special Educational Needs (SEN), particularly those with Autism, the use of a computer can even help to reduce anxiety and effectively aid communication (Autism Education Trust, 2009).

Over to you

So, how can you provide more opportunities for pupils to talk and collaborate when teaching computing? Do let me know via Twitter. You can find me @computingchamps. (HW)
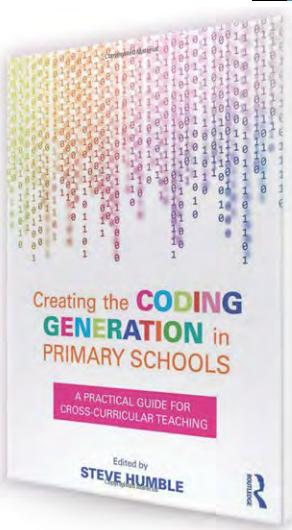
Neil is a Senior Lecturer in Computing Education at the University of Hertfordshire. He is a local CAS community leader, and a Raspberry Pi, Google, and Microsoft Certified Educator.

CREATING THE CODING GENERATION IN PRIMARY SCHOOLS

A set of sometimes provocative, always interesting, perspectives on primary computer science education that secondary teachers might also find useful

Humble has put together an interesting, informative, and accessible collection of articles exploring CS education as it might take place in primary schools. However, just as there's more to CS education than coding, there's more to this book than a focus on teaching a generation to code, and there's much here that will be of interest to those working outside of primary education.

The chapters are divided into three sections: teaching coding, which addresses some of the broader contextual issues around CS education, the subject of coding, addressing some specific pedagogical approaches, with illustrative examples in a number of languages, and a short final section on coding and the wider curriculum, looking at uses of the Raspberry Pi, and touching on the relationship between computational thinking and coding, the transition from blocks to text in programming, and the place of creativity and collaboration in coding.

Bell, Duncan, and Rainer's opening chapter reflects on the increasing use of 'coding' as a buzz word, as the title of the book itself reflects. The authors helpfully draw our attention to all that this might encompass: programming, the broader discipline of computer science, computational thinking, and the, perhaps rather neglected at school level, discipline of software engineering. Bell et *al* illustrate both programming and computational thinking through the example of check digits in barcodes, effectively emphasising the connections between computational thinking and coding. The chapter includes a number of 'reflection activities', where the authors pose questions for their readers, such as: "Should we teach programming so that pupils can do computer science, or computer science so that pupils can do programming?"

It's no surprise that computational thinking is a theme running throughout the book, and Walker and Gleaves's chapter provides a useful overview of the topic, and how it's addressed through the English national curriculum. While acknowledging other ways in which computational thinking is framed, they identify three concepts at its core – audacity, abstraction, and automation – although I'm not sure that some of their suggestions for teaching computational thinking (e.g. taking a dog for a walk, recognising repeating melodies, or setting up equipment for a sports game) remain entirely true to this model. The creative tension around the extent to which computational thinking should result in computation isn't really addressed.

In their provocative chapter on considerations for teaching coding to children, Stager and Martinez give us a well thought through set of principles to inform our teaching, taking an unashamedly tech-based approach to the teaching of computer science, for example: "there is no computer science without computers", "physical computing is a critical context for learning computer science", and "there is no substitute for personal computing".

Cross-curricular applications of coding, and computational thinking, are another theme, and perhaps do make the book more relevant to primary practice. We've coding examples for mathematics (in BASIC) from Humble himself, and music by Aaron using Sonic Pi, but the pedagogic insights from Mitra and colleagues on self-organising learning environments, Burke on DIY design in Scratch, and Liao on coder poets, amongst others, lend themselves to application beyond mere coding.

The articles have, I suspect, been assembled with an audience of primary trainee teachers in mind, but there's much to commend the book to more experienced teachers of primary and secondary computing. A book like this struggles to offer a coherent argument, nor does it attempt to teach anyone how to code, but it offers an illuminating and engaging set of quite different perspectives on coding and its broader curriculum context. You may not agree with everything, but even the bits which don't ring true for you are likely to still give a productive pause for thought. **(HW)**

CODIERTE KUNST: KUNST PROGRAMMIEREN MIT SNAP! (CODED ART: ART PROGRAMMING WITH SNAP!)

IN FO **BY** Joachim Wedekind | **PUBLISHER** Joachim Wedekind | **PRICE** €34.80 + €10.60 postage, e-book (PDF): €22.00 | **ISBN** 9780262534307 | **URL** helloworld.cc/2C2bH2b

The author asks the question 'Is it possible to learn programming and to create aesthetic graphical objects right from the start?' This is a lovely book, in content and as an object itself, using Snap! to create homages to early examples of computer art, and to examples from conceptual and op art.

The text of the book is in German (the link given is to an English description of the book), but the programs are easily translatable (it's easy to use Snap! in German or another 40 languages including non-Roman alphabet and right-to-left languages) and all are available on the website for download.

Programming concepts and user-defined functions are introduced whenever needed for the recoding of early computer art, and then used in remixing, altering, and extending these examples as well as the development of new works.

Even if you don't get the book (leave it on an art teacher's desk and spark some discussion/collaboration!) use the author's websites as an inspiration. **(HW)**

GENERATIVE DESIGN: VISUALIZE, PROGRAM, AND CREATE WITH PROCESSING

IN FO **BY** Hartmut Bohnacker, Benedikt Groß, and Julia Laub | **PUBLISHER** Princeton Architectural Press | **PRICE** £39.00 (Amazon) | **ISBN** 978-1616890773 **URL** helloworld.cc/2R5y94V and vimeo.com/15658375 (flipbook)

Another wonderful book from Germany (this one's been translated into English) on the intersection between art/design and programming, this time with the emphasis on generative design, where altering the parameters or algorithm of a generating process can produce an almost unlimited number of solutions and the designer's job is to pick 'the best'.

It uses Processing rather than Snap!, although a new version of the book due in October uses the p5.js library rather than Processing itself, which might be easier in a school setting.

The book is beautifully produced, starting with a really useful image overview of the entire book, a How to read this book double spread, and (like *Codierte Kunst*) glossy full-page spreads showing design examples, but after these the programming starts, with Basic Principles (colour, shape, type, and images), followed by Complex Methods (randomness, oscillation, formulated bodies, attractors, tree diagrams, and dynamic data structures).

This book is a complementary one to *Codierte Kunst*, but the pair would make really useful resources for an art/design/programming course. **(HW)**

ESSENTIAL READING

Some great Scratch programming titles - there's lots more to this than moving cats across screens!

CODE CLUB BOOK OF SCRATCH, VOLUME 1

BY Rik Cross and Tracy Gardner
PUBLISHER Raspberry Pi Press
PRICE Free (PDF) or £9.99 in print
ISBN 978-1912047673
URL helloworld.cc/2GRqwdC

A great 'getting started' guide to Scratch from the Code Club team, with copious illustrations, step-by-step instructions, and plenty of ideas for taking the six featured projects further.

COMPUTER SCIENCE CONCEPTS IN SCRATCH
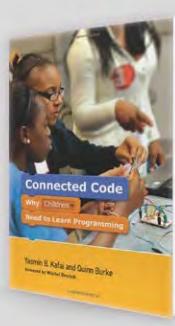
BY Michal Armoni and Moti Ben-Ari
PUBLISHER Weizmann Institute of Science
PRICE Free
URL helloworld.cc/2Rxak5n

Much more about learning computer science through the medium of Scratch than learning to code in Scratch, although plenty of Scratch coding will be picked up through solving the exercises presented here!

CONNECTED CODE: WHY CHILDREN NEED TO LEARN PROGRAMMING

BY Yasmin Kafai and Quinn Burke
PUBLISHER The MIT Press
PRICE £24
ISBN 978-0262027755
URL helloworld.cc/2VvGkpy

This particular release is in part a history of Scratch itself, and in part a well-argued case for a pedagogy that emphasises computational participation over computational thinking: making and sharing matter!

Q WILL THERE BE AN OFFLINE VERSION OF SCRATCH 3 TO INSTALL [ON OUR OLD LAPTOPS]?

A We kick off this issue's collection of FAQs with one we've been asked a lot, relating to the release of Scratch 3. To answer this first collection of questions, we've brought in Katherine from the Code Club team, who has been working towards the release of Scratch 3 for some time now.

Our understanding is that an offline version of Scratch 3 will be made available. Furthermore, you'll be able to download and install it on laptops or computers from January 2019 (around the time this issue of the magazine is landing with you, in fact).

We're working closely with the MIT Scratch team during this time of transition and will continue to update clubs with more information as we progress towards that release date.

Q WILL ALL THE CODE CLUB PROJECTS BE UPDATED AND READY IN TIME FOR THE LAUNCH OF SCRATCH 3?

A We're very excited about the launch of Scratch 3, as you might expect. As such, we've been working very hard in advance of its release to ensure that Scratch projects will be updated to include instructions for Scratch 3 alongside Scratch 2 by 2 January 2019. That means, again, that by the time you have this magazine in your hands, that work should have been completed and you'll be able to see that available projects have already been updated for Scratch 3.

In time for the launch, we aim to have Scratch 3 versions of Code Club modules 1-3 available – these should be ready for you to start using now.

Q ALL OUR PCS USE INTERNET EXPLORER AND WE CAN'T UPGRADE TO EDGE, INSTALL CHROME OR DOWNLOAD AN OFFLINE VERSION OF SCRATCH. WHAT CAN WE DO?

A The requirement for computers to run the Microsoft Edge web browser instead of Internet Explorer is one that has been decided by the team at MIT in America, who develop and maintain Scratch.

We recognise that this might mean that some clubs need to update some of their software. In advance of that, we've been communicating this change to clubs for the past few months to give everyone time to plan ahead and make necessary changes.

We recommend that you discuss your own software requirements with your IT support team as soon as possible. If we can provide any support with specifications or minimum requirements, please let us know.
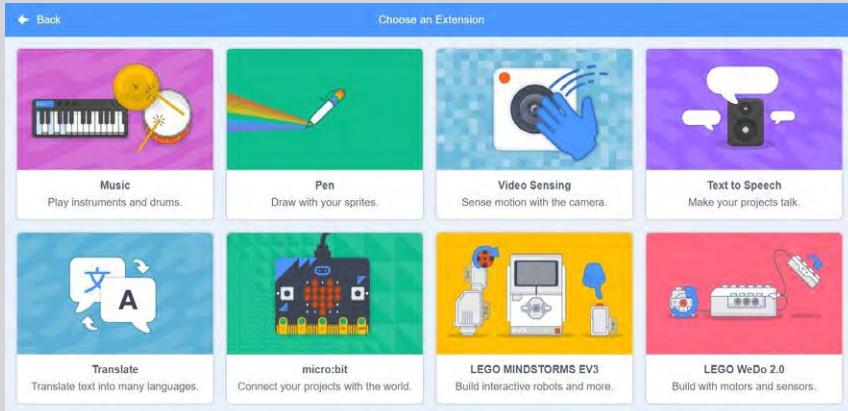
Q WHO DO WE REQUEST/REPORT PROJECT VARIANCES TO?

**A** The team at Code Club very much welcome user feedback, and there are a couple of ways that you can send this over to them.

Feedback can be left at the end of the Code Club project page over at **helloworld.cc/2Runtfy.** Alternatively, you can email your feedback and/or queries if you prefer. You need to send them to **support@codeclub.org** where they'll then be passed on to the relevant team.

## Q ARE THERE ANY DIFFERENCES WHICH MIGHT CAUSE ME PROBLEMS WHEN USING SCRATCH 3?

A Yes, some of the blocks such as the pen have been moved into extensions, so you'll need to add the extension if you want to use them. There have also been some small changes elsewhere, particularly to lists where you can no longer select a random item or the last item from a list without additional blocks. However, there are also some other new blocks which will save you time – for example, there's now a block which lets you get the current costume name rather than just the number. The colour selector is also different, opting for a hue, saturation, and brightness approach, so creating colours such as white and black might take a little getting used to.

Q I WANT TO TRY SOMETHING DIFFERENT IN MY CLASSROOM. HAVE YOU GOT ANY SUGGESTIONS FOR USEFUL SOFTWARE OR WEBSITES I MAY NOT HAVE SEEN OR USED BEFORE?

A If you want to link hardware devices to online services (Twitter glowing orb, anyone?) then why not check out Node Red? It's very easy to install on a Raspberry Pi and lets you put together a visual 'flow' of information involving inputs, processes, and outputs. Try out this tutorial if that sounds interesting, and you want to get started: **helloworld. cc/2AxgCs1**

Q IS THERE STILL A VERSION OF SCRATCH 2 AVAILABLE TO DOWNLOAD AND INSTALL?

A Just because there's a new version of Scratch released, that doesn't mean the older one has disappeared off the face of the planet. As such, yes, there is a downloadable version of Scratch 2. If you want to get hold of it, you can find it on the Scratch website here: **helloworld.cc/2LOIU5G**

That said, we do advise clubs to plan to upgrade to Scratch 3 over the coming year. By doing so, you'll be able to make use of the new features and functionality available with the latest version of Scratch. To get the most out of it, it's very much worth making sure you have the latest version up and running.

Q MY STUDENTS HATE PLANNING PROGRAMS. HOW CAN I GET THEM TO STOP AVOIDING THIS STEP?

A One way of planning a program is with a flow chart, and a quick and easy way of prototyping a flow chart might help. A useful site is **www.draw.io** which makes it easy to draw, modify, and save flow charts.

Q WILL SCRATCH 3 WORK ON THE RASPBERRY PI?

A Yes it will, but not immediately. Following the release of Scratch 3 on 2 January 2019, we started work on a version of it for the Raspberry Pi. We're working closely with the MIT Scratch team on the Raspberry Pi version, and we expect this to be available in the second quarter of the year. You should see it by the summer.

Q WHAT HAPPENS TO ALL MY SCRATCH 2 PROJECTS?

A **For online users**: Your projects will be upgraded to the new Scratch 3 format automatically.

For offline users: All your saved projects can be opened in Scratch 3 and saved in Scratch 3 format, provided you've downloaded and installed Scratch 3 on your computer.

Q WILL THERE BE A SCRATCH 3 APP FOR TABLETS?

A Scratch 3 will be accessed via a browser on both iPads and Android browsers.

The Code Club team is working closely with the MIT Scratch team during this transition and will continue to update clubs with more information as and when it becomes available.

Q IF I USE SCRATCH 3 ON A TABLET, IS THERE ANYTHING I CAN'T DO?

A Right-click context menus (such as deleting or duplicating a sprite) won't be implemented on tablets until later in 2019, but there are usually ways of achieving the same goal without needing to right click. Similarly, although you can use keyboard input with the input block, you can't use the 'key pressed' blocks with the onscreen keyboard, a feature which MIT intends to implement later in the year.

Q WHEN WILL SCRATCH DESKTOP BE AVAILABLE FOR CHROMEBOOKS?

A Scratch Desktop for Chromebooks isn't yet available. The MIT Scratch team are working on it and expect to release it later in 2019.

YOUR LETTERS

Our letters page is a place for you to join our conversation. If you've got a comment, a question or an announcement to share, contact us on Twitter via **@helloWorld_Edu** or using the **#HelloWorld** hashtag. Alternatively, email us with 'Teacher Letter' in the subject line (**contact@helloworld.cc**).

Hands On

Dear Hello World,

Thank you so much for the articles on incorporating Minecraft into lesson ideas in issue six of *Hello World*. Even though most of my students appeared to have moved onto Fortnite in recent months, we loved the idea of a bicycle tour in Minecraft, and it helped bring some of those who considered themselves 'too old' for Minecraft back to the game!

I always look for ideas such as these, though. Something that can take lessons towards something practical, that the class of potentially otherwise bored students instantly buys into.

I wonder if you can explore other ideas, and perhaps give us a few more Minecraft-themed things we can try, too. But anything that allows me to mix the Raspberry Pi and something to save the first five minutes of a lesson, where I have to try my damnedest to get students fully invested in what we're doing!

Keep up the good work, and thank you for the magazine,
Peter Morris, London

Archive

Dear Hello World,

Thanks for the heads-up in the last issue in the direction of the BBC Computer Literary Archive (helloworld. cc/2Py6FQ7). I wasn't aware that all of this historical material had been made available online, and what a treasure trove it's proven to be! Fascinating, too, to read that the Raspberry Pi itself was inspired by the BBC Micro!

It's just a shame that the BBC drama *Micro Men* has never, to my knowledge, been released on DVD. This was from back in 2009, and featured Alexander Armstrong as Sir Clive Sinclair and Martin Freeman as Chris Curry. A rare attempt by British television makers to examine the history of computing in an accessible drama. We see it with history and politics a lot, but it'd be a lovely resource to have. Regrettably, I didn't record *Micro Men* when it was first on, and so my students have missed out on the pleasure of it. No worries: we've got the BBC archive to work through now!
T Osborne

We remember that programme well! Hard to believe that it has been nearly ten years since it was on, too. The official website for it is still up and running (**helloworld.cc/2CTETu9**), and, according to that, it hasn't been repeated since 2011. It just won't do!

The programme itself wasn't perfect if our memory of it serves, but it was a genuine attempt to document an amazing era in British computing history. More dramas of its ilk would be very welcome. We hope, instead, your students are enjoying the aforementioned BBC archive.

Many thanks for getting in touch, Peter, and you'll hopefully be pleased to know that we do indeed have more Minecraft in this issue! You'll find that over on p38. You'll also find ideas surrounding making an animated film, getting students involved with photography, and sending programs into space.

However, as we always say, if there's something you'd like to see in the magazine that we haven't already covered, then please do get in touch. We'd love to hear from you. Our contact details are at the top of this page.

More World?

Dear Hello World,

A colleague passed over to me the latest edition of *Hello World* magazine (issue 6) and I've signed up to subscribe pretty much straight away thereafter! I did wonder though if you have any back issues of the magazine available, so us latecomers can catch up with what we may have missed?
Ann

Thanks for signing up for a subscription, Ann. Hopefully this is good news, too: you can download every issue of the magazine today absolutely free of charge from our website. You'll find the full archive at **www.helloworld.cc**. We hope you find plenty in there that's of use.

Sorting Out Scratch

Dear Hello World,

Issue five of *Hello World* talked about the incoming Scratch 3, and of course it's due with us very soon. Please could I make a request, then, for *Hello World*'s team of experts to – on top of the ideas they kindly share with us – give us a basic overview of what it is we need to know about Scratch 3 in the first place? Ideally something that can be shared with colleagues less familiar with it. I'm already being asked if Scratch 3 will make Scratch 2 obsolete, for instance, and if it's going to require heavy investment to deploy. For those of us in smaller institutions, without perhaps the funding and breadth of expertise of bigger places, any material you can provide along those lines would be very helpful indeed.
Mr N K Hardy, Sheffield

Thank you for writing in, Mr Hardy, and as you can see, this issue of *Hello World* is very Scratch-centric! In terms of the core kind of questions you're asking, we've brought a collection of answers together in this issue's FAQ section, just a few pages before these letters. Head to p90, and hopefully the kind of answers you were looking for are right there.

BUDGET TROUBLES

Dear Hello World,

Across the last two issues of *Hello World*, you've printed a pair of letters from correspondents who have supplied their name and asked for it not to be printed. I fully understand the reasons for them doing this, given that they were unhappy at budget levels, and the increasing paradox between what they're expected to do, and the amount of available funding to do it.

If I may, though: it's a worrying state of affairs that they feel they have to withhold their names, for fear presumably of comeback and consequences. Yet their unhappiness is reflected, anecdotally, by colleagues across the CS teaching profession. Is there a way, I wonder, that these conversations can be channelled without people feeling they are in some way endangering their professional position for having such chats in the first place?

Keep up the good work,
A R Matthewson, Cardiff

A fair question, and thank you for raising it. One thing that unites pretty much everybody we've met when it comes to teaching CS is a desire and drive to give students the absolute best, oftentimes against a difficult backdrop of tightening resources. Our letters pages each issue remain very much open if more of you want to continue this discussion. **(HW)**

We'll print another selection of your correspondence in *Hello World* 8. Until then, as always, thanks to everyone who's got in touch.

MILES BERRY PRINCIPAL LECTURER

IN PRAISE OF BLOCKS

Why we should resist the urge to move on to teaching a text-based language too soon

Scratch may not have been the first block-based programming language, nor is it the most flexible, but it's undoubtedly the world's most popular. I don't think it's an exaggeration to suggest that the global movement to teach children to program wouldn't have got very far if it hadn't been for Scratch or something very much like it. I think there's a danger that CS educators working with upper primary or lower secondary students often want to move on to Python or another text-based language sooner rather than later, in part because they can, but also perhaps because pupils moan that they've 'done Scratch'. I think it's worth resisting this urge, staying a little bit longer with Scratch, and making sure the foundations of coding and computational thinking are really firm, for all our pupils, before we subject them to the additional cognitive load of text-based programming. Let me give you some of the reasons why.

Programming is *hard*, but text-based programming is particularly difficult because the programmer has to think about two things at the same time: the semantics of the program, the algorithms and data structures that the programmer is trying to code, and the syntax of the program, the particular vocabulary, grammar, and punctuation that's needed to express it in Python or whatever. In Scratch, the syntax is taken care of by the blocks provided in the language, and thus the programmer gets to focus much more of their thinking on the semantics. If what we're interested in is developing pupils' 'computational thinking', then Scratch makes it much easier to emphasise this, rather than having to master the syntax of Python.

One consequence of the syntax being built into the blocks is that it's all but impossible to make syntax errors when using a visual language. Many of the frustrations felt by those learning to program in text-based languages are due to the frequency with which they'll make syntax errors in their code, thus visual language programs are far less likely to have errors than their text-based equivalents, and the errors that programmers do make are likely to be much more *interesting*, providing much more opportunity for learning from mistakes than simple typos.

In developing Scratch, Resnick and his team were deeply inspired by construction toys, from Froebel's gifts through to modern LEGO. While LEGO sets typically come with step-by-step instructions, younger children will just *play* with LEGO, making whatever they have in mind through clicking blocks together and seeing what works, and more experienced LEGO builders will range far beyond any step-by-step guides, again letting their imagination take the lead in creating novel, original artefacts. Because it's so easy to playfully experiment in Scratch, pupils can quickly try ideas out to see what difference they make, and to figure out for themselves what works, or what improves their program, and what doesn't.

By making it easy to learn the programming language, Scratch makes it possible for young programmers to acquire a sense of mastery of the tool, of fluency in the language. After an introduction to the language, and plenty of opportunity for play and experiment, many children pass beyond learning to code to being able to express their creative ideas fluently through the medium of code: instead of spending time learning the language, they can spend their time building, making, experimenting, and sharing. **(HW)**

Miles is Principal Lecturer in computing education at the University of Roehampton. He's a member of the Raspberry Pi Foundation and serves on the boards of CAS and CSTA.

"HELLO, WORLD!"

Everything you need to know about our computing and digital making magazine for educators

Q WHAT IS HELLO WORLD?

A Hello World magazine is a magazine for computing and digital making educators. Written by educators, for educators, the magazine is designed as a platform to help you find inspiration, share experiences, and learn from each other.

Q WHO MAKES HELLO WORLD?

A The magazine is a joint collaboration between its publisher, Raspberry Pi, and Computing At School (part of BCS, The Chartered Institute for IT). Hello World is sponsored by BT.

Q WHY DID WE MAKE IT?

A There's growing momentum behind the idea of putting computing and digital making at the heart of modern education, and we feel there's a need to do more to connect with and support educators inside and outside the classroom.

Q WHEN IS IT AVAILABLE?

A Your 100-page magazine will be available three times per year in time for each new term in January, April, and September. Would you like it to be available more frequently? Let us know!

IT'S FREE!

Hello World is free now and forever as a Creative Commons PDF download. You can download every issue from **helloworld.cc**. Visit the site to see if you're entitled to a free print edition, too.

WANT TO GET INVOLVED?

There are numerous ways for you to get involved with the magazine.
Here are just a handful of ideas to get you started:

- **Give us feedback**
  Help us make your magazine better –
  your feedback is greatly appreciated.

- **Ask us a question**
  Do you have a question for a FAQ section or a
  bugbear you'd like to share? We'll feature your
  thoughts and ideas.

- **Tell us your story**
  Have you had a recent success (or failure) you
  think the wider community would benefit from
  hearing? We'd like to share it.

- **Write for the magazine**
  Do you have an interesting article idea?
  We'd love to hear from you.

GET IN TOUCH

Want to talk? You can reach us at:
contact@helloworld.cc

FIND US ONLINE

www.helloworld.cc

🐦 @HelloWorld_Edu

f fb.com/HelloWorldEduMag

SUBSCRIBE IN PRINT TODAY!
PAGES 32-33

(Hello World)

helloworld.cc