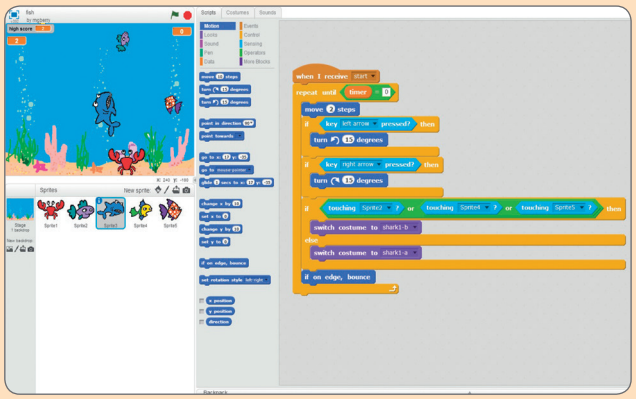
UNIT 5.1

We are game developers

Developing an interactive game

1 About this unit

Software: Scratch (or Kodu)
Apps: Snap! in the web browser (Scratch requires Flash, which is not available on iPad)
Hardware: Desktop/laptop computers, microphones
Outcome: An original computer game, ideally uploaded to the Scratch community site



UNIT SUMMARY

The pupils plan their own simple computer game. They design characters and backgrounds, and create a working prototype, which they develop further based on feedback they receive.

CURRICULUM LINKS

Computing PoS

- Design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts.
- Use sequence, selection, and repetition in programs; work with variables and various forms of input and output.
- Use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs.
- Select, use and combine a variety of software (including internet services) on a range of digital devices to design and create a range of programs, systems and content that accomplish given goals ...

Suggested subject links

- **Art and design:** Pupils can improve their art and design skills by creating artwork for their games.
- **Music:** Pupils can record sound or compose music for their games.
- The games may require an understanding of aspects of maths and science to ensure the computer model on which the game is based is realistic.

TRANSLATING THE COMPUTING PoS

- Making a computer game gives ample scope for pupils to *design and create programs to accomplish a given goal*.
- The pupils will be working with a *variety of input and output*, which will include keyboard and/or mouse (*input*), and the computer display together with speakers or headphones (*output*).
- Creating the games involves common programming constructs such as *sequences* of instructions, *selection* (the behaviour of the game varies according to the player's actions) and *repetition* (which might be dependent on a particular event occurring, such as clicking a sprite).
- If games use scores, levels, randomisation or time limits, the pupils will need to work with *variables*.
- The pupils' games are unlikely to work first time, so they'll need to *use logical reasoning to detect and correct errors*. As they provide feedback to one another, they'll become more *discerning in evaluating digital content*.

LEARNING EXPECTATIONS

This unit will enable the children to:

- create original artwork and sound for a game
- design and create a computer program for a computer game, which uses sequence, selection, repetition and variables
- detect and correct errors in their computer game
- use iterative development techniques (making and testing a series of small changes) to improve their game.

The assessment guidance on page 20 will help you to decide whether the children have met these expectations.

VARIATIONS TO TRY

- Scratch is strongly recommended, but other game development toolkits are available. Kodu provides a rich, immersive 3D environment.
- The MaKey MaKey controller offers an alternative approach to providing keyboard input. See www.makeymakey.com.

2 Getting ready

THINGS TO DO

- The work in this unit assumes the pupils have completed earlier programming and computational thinking units in *Switched on Computing*; if they have not, they may need additional time and support to create an enjoyable, playable game.
- Read the *Core steps* sections of *Running the task*.
- Decide which software/tools are most accessible/appropriate for use with your class.
- Download your chosen software/tools (see *Useful links*) and spend some time familiarising yourself with them.
- Watch the *Software in 60 seconds* walkthroughs for this unit.
- Think about the individuals and groups you have in your class. Could you use any of the *Extensions* on pages 14–19 to extend your more able children? Could you use any of the suggestions in *Inclusion* (see below) to support children with specific needs, e.g. SEN or EAL? Have you considered how a Teaching Assistant will support you and the children, if one is available?
- Ensure you have sufficient computers/laptops/tablets and other equipment booked in advance.

THINGS YOU NEED

- Relevant exemplification from the web (see *Useful links*)
- Computers with internet access
- Microphones
- Ideas of curriculum topics for games, if desired

CD-ROM RESOURCES

- Example games written in Scratch (also available online)
- Unit poster – How we program
- *Software in 60 seconds* – Introduction to Snap!
- *Software in 60 seconds* – Scratch (2–7)
- Storyboard templates
- Pupil self-assessment information

E-SAFETY

- Pupils don't need accounts to download Scratch 1.4 or Scratch 2.0, or to use Scratch 2.0 or Snap! online.
- If the pupils do register for accounts, they need to give a parent's or carer's email address, for which you will need permission.
- Once registered, the pupils can share their work with the global Scratch community in a safe online space.
- You might allow the pupils to incorporate downloaded images and sound effects into their games, but you should respect any licence conditions and intellectual property rights, and ensure the usual precautions for safe searching are in place.

INCLUSION

- Scratch has built-in support for a number of languages.
- Ask the pupils to think about inclusion and accessibility as they develop their game, e.g. by providing audio and visual prompts for questions.
- Programming makes considerable demands on the pupils' thinking, and some pupils may need additional support. Most pupils will do best working with a partner.

USEFUL LINKS

Software and tools

- Scratch is free software. Download from http://scratch.mit.edu/scratch_1.4 or use online at http://scratch.mit.edu/projects/editor.
- Snap! is free open source software. Use online at http://snap.berkeley.edu/snapsource/snap.html.
- Kodu is free software that can be installed on modern Windows computers: www.kodugamelab.com.

Online tutorials

- Introduction to Scratch 2.0: http://info.scratch.mit.edu/Video_Tutorials.
- Interactive tutorials for Kodu are included in the download.

Information and ideas

- Scratch: http://scratch.mit.edu.
- Scratch educator community: http://scratched.media.mit.edu.
- Code Club materials for the 'Whack-a-Witch' game: http://codeclub-assets.s3.amazonaws.com/public/codeclub-whackawitch.pdf.
- Snap!: http://byob.berkeley.edu.
- Flickr: www.flickr.com.
- Freesound: www.freesound.org.
- Rubber duck debugging: http://en.wikipedia.org/wiki/Rubber_duck_debugging.

Games

- Angry Birds: http://chrome.angrybirds.com.
- Light-bot: http://light-bot.com/flash-lite.html.
- Doctor Who 50th Anniversary game: www.google.com/doodles/doctor-whos-50th-anniversary.
- Some simple Scratch games, e.g. http://scratch.mit.edu/projects/15906446, http://scratch.mit.edu/projects/15906870, http://scratch.mit.edu/projects/15907506.

3 Running the task – We are game developers

Software: Scratch (or Kodu) **Apps:** Snap! in the web browser (Scratch requires Flash, which is not available on iPad) **Hardware:** Desktop/laptop computers, microphones
Outcome: An original computer game, ideally uploaded to the Scratch community site

Core steps

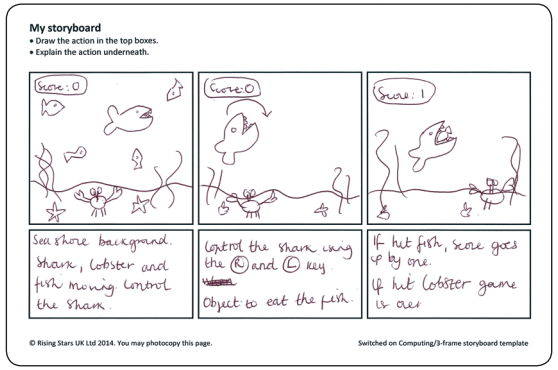
Step 1: Planning a game

RESOURCES

- Storyboard templates

Examples of online games:
- http://chrome.angrybirds.com
- http://light-bot.com/flash-lite.html
- www.google.com/doodles/doctor-whos-50th-anniversary

POSSIBLE OUTCOME FOR THIS STEP:



- Begin by asking the pupils to discuss the computer games they play. Ask them to describe the algorithms these games are based on. Show the pupils an example of an online game (see *Resources*). Ask them to identify what makes some games particularly enjoyable to play. You might like to use the characteristics they identify as success criteria for the games they will be creating in this unit.
- Tell the pupils they are going to create their own games. Assign each pupil a partner (pairing pupils is highly recommended for this activity).
- Ask the pupils to think about the structure of their game. What are the objectives? What sort of player is it intended for? What will happen in the game? Is there any progression built in? It may be useful for the pupils to write down these ideas. Encourage them to keep their ideas simple. Explain that it's better to create a simple, working game than to become frustrated because their ideas exceed their programming knowledge.
- Ask the pupils to record the algorithm for their game as a series of scenes on a storyboard, as a flow chart or as a story.
- Ask the pupils to give each other feedback on their game ideas and algorithms. Review the pupils' ideas and provide feedback yourself. Give the pupils the opportunity to revise their plans and algorithms.

Extensions

SCHOOL

- Some pupils could use digital tools to draw a flow chart or storyboard for their game (e.g. Microsoft PowerPoint®), or to sketch characters and backgrounds (e.g. Microsoft Paint, IWB software or the Brushes app).

HOME

- Encourage the pupils to share their game ideas, including the initial algorithm, with their parents or carers. Ask them to revise their plans in the light of the feedback they receive.

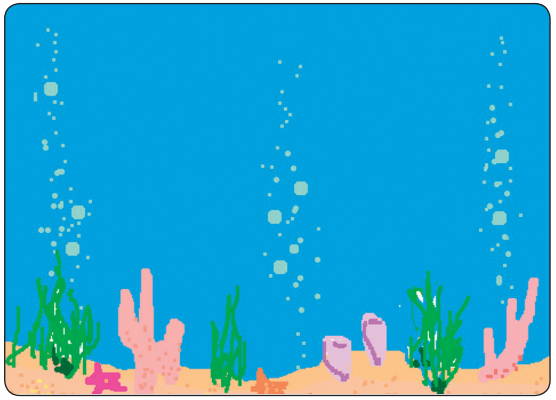
Step 2: Creating and sourcing assets

RESOURCES

- *Software in 60 seconds* – Scratch (2, 3, 4)

- Flickr: www.flickr.com
- Freesound: www.freesound.org

POSSIBLE OUTCOME FOR THIS STEP:



- Ask the pupils to think about the assets they'll need for their games, such as backgrounds, costumes, music and sound effects. Encourage them to sketch some of their ideas.
- Remind the pupils how the background and sprite editor works in Scratch, or ask one or more pupils to demonstrate this to the class. Discuss the difference between bitmap and vector modes in the editor – in bitmap mode, each dot (pixel) is specified, whereas in vector mode, the lines and other shapes are defined precisely.
- Set the pupils the challenge of creating the backgrounds and sprites they need for their game.
- Rather than creating game assets themselves, the pupils could source them from Creative Commons licensed content, using websites such as Flickr or Freesound (see *Resources*).
- Encourage the pupils to create multiple costumes for one or more of their sprites, either to allow more realistic movement using animation techniques, or to include actions according to the ideas they're incorporating into their game. Ask them how they might achieve this (if necessary, remind them that they can do this by switching costumes).
- Demonstrate how the sound recorder/editor works in Scratch, or ask one or more pupils to demonstrate.
- Set the pupils the challenge of recording suitable sound effects and dialogue for their games.
- The pupils might also like to record backing music for their games. One way of doing this is to use the *music* blocks in Scratch. The advantage of using Scratch's built-in tools to create backing music over simply importing a sound file is that the tempo, pitch and volume of the music can be altered according to events in the game.
- Ask the pupils to show their work to one another, making changes in response to the feedback they receive.

SCHOOL

- Some pupils could use other digital tools to create artwork, sound and music, saving them in standard formats and then importing them into Scratch.

HOME

- The pupils can continue to develop their game assets at home, either in Scratch or using other digital tools.

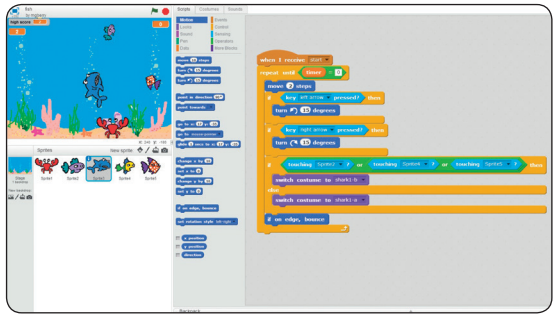
Core steps

Step 3: Creating a prototype of the game

RESOURCES

- Examples of games written in Scratch, such as http://scratch.mit.edu/projects/15906446, http://scratch.mit.edu/projects/15906870 and http://scratch.mit.edu/projects/15907506

POSSIBLE OUTCOME FOR THIS STEP:



- Remind the pupils how the Scratch script editor works, or ask one or more pupils to demonstrate this to the class.
- Before the pupils start programming their games, it's worth showing them a few ideas for how to program some simple games, such as chasing sprites, steering a car, escaping from a maze and/or shooting a target. Some examples can be found online – see *Resources*.
- Model the process of thinking through an algorithm and then coding it using Scratch's blocks. The pupils should have their own algorithms, at least in outline form, which they can then translate into Scratch *command* blocks.
- Encourage the pupils to solve the problem of creating their game by breaking it down into its component parts and then solving each part. They should keep in mind the overall game while they do this. Their outline algorithms should help them.
- The pupils might start by thinking about how they will control the movement of the player's sprite, typically using the keyboard or the mouse, with a *repeat* block and *point towards mouse* or *when key pressed* blocks. The pupils should also think about how they will control other elements of their game, such as sprites to be chased, obstacles and rewards. They may find it helpful to use the *pick random* block for these elements.
- The pupils will need to think about how the player's sprite interacts with other elements in the game: this is likely to involve *if/then/else* blocks, *sensing* blocks, such as *touching*, or perhaps *broadcast* and *when I receive events*.
- The pupils should include some way of tracking progress or building in challenge to their game, perhaps through scores, lives or a countdown timer. Demonstrate how they can use variables to do this.

Extensions

SCHOOL

- There's plenty of scope in Scratch for the pupils to extend their programming skills beyond the basics. Using the *Make a block* button in Scratch's *More Blocks* palette would allow the pupils to simplify their code and gain experience in creating procedures.

HOME

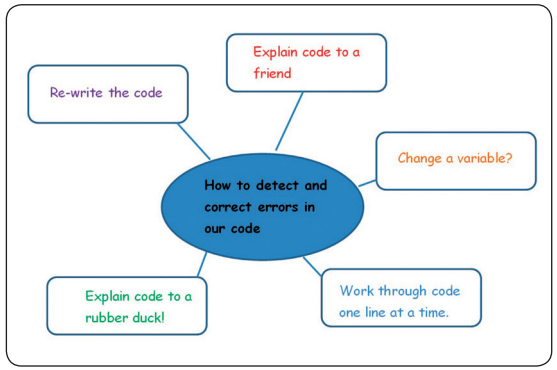
- Encourage the pupils to continue to work on their scripts at home. Even if their parents or carers can't help with the programming, having someone to explain their programs to will develop their logical thinking.

Step 4: Debugging the game script

RESOURCES

- Rubber duck debugging: http://en.wikipedia.org/wiki/Rubber_duck_debugging

POSSIBLE OUTCOME FOR THIS STEP:



- It's unlikely that the pupils' Scratch scripts will work as planned the first time they're run. This isn't a problem – the pupils will learn a lot about programming, and computational thinking, through the process of debugging their programs.
- Remind the pupils of their earlier experience of debugging programs in *Unit 3.2 – We are bug fixers* and some of the other programming units in *Switched on Computing*. Draw a parallel with playing computer games, where it's important to persevere in order to solve a puzzle or complete a level.
- Brainstorm techniques the pupils can use to detect and correct errors in their code, such as explaining to a friend (or even a rubber duck – see *Resources*) what the code does, isolating the bit of the code that is causing the problem (perhaps by stepping through the program one line at a time), changing variables, or rewriting code. Emphasise the importance of using logical thinking in this process, and of the scientific method of changing just one input (or line of code) to see whether outputs change as predicted.
- Remind the pupils of different types of bug that can occur, emphasising the difference between bugs in their algorithms (where their approach to solving the problem is wrong) and bugs in the implementation of their algorithms (where they've not correctly converted their ideas into Scratch code).
- Support the pupils as they work to fix their programs. Consider assigning the pupils a debugging partner, even if they are working individually, with each helping the other overcome the problems in their code.
- It's possible the pupils will encounter problems that are caused by limitations inherent in Scratch itself – in these cases, encourage the pupils to think of ways of working around the problem to avoid the limitation.

SCHOOL

- The pupils could post their buggy code to the Scratch community site, requesting help from more experienced users, or you could do this on their behalf.
- The Scratch community forums might also provide useful information about ways of solving some of the problems the pupils encounter.

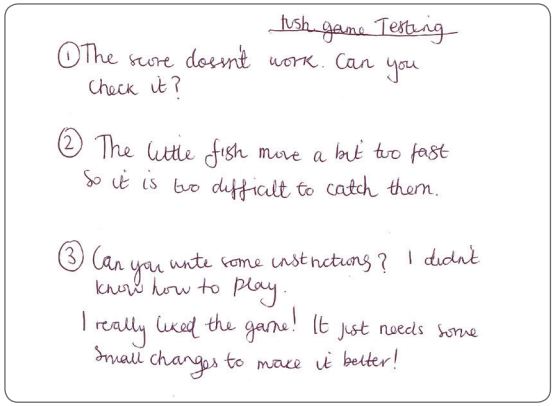
HOME

- Encourage the pupils to continue to debug their code at home. Explaining the problem to a family member can be an effective way of finding a solution, as can returning to the problem after a break.

Core steps

Step 5: Testing the game

POSSIBLE OUTCOME FOR THIS STEP:



- Remind the pupils of the criteria they arrived at in Step 1 for what makes a good computer game. Encourage the pupils to look at their games with a critical eye, using these criteria to guide them. For example, how easy is the game for a beginner to play? Does it offer sufficient challenge to make someone want to play it more than once? Are the graphics and sound as effective as they could be?
- Give the pupils time to work on their games to improve their playability.
- Explain that the key to creating a good game is testing it on users, and developing it further on the basis of their feedback.
- Assign each pupil (or pair) a partner (or partnering pair) to test and provide feedback on the game. Encourage the pupils to give each other specific suggestions for how the game can be improved, writing their suggestions down so that the original programmers can work through this list.
- Support the pupils as they work through the suggestions for improvements, addressing as many of them as they can in the time available. Remind the pupils that they'll need to test their code carefully to ensure that any changes and new features work properly.
- If time allows, assign the pupils/pairs to another partner/partnering pair and repeat the process to get further feedback, which the programmers can use to make further changes to their games.

Extensions

SCHOOL

- Some pupils could post their games to the Scratch community site to get feedback from other members of the community.

HOME

- Encourage the pupils to show their games to their parents, carers or siblings, asking for additional feedback, and developing their games further on the basis of the feedback they receive.

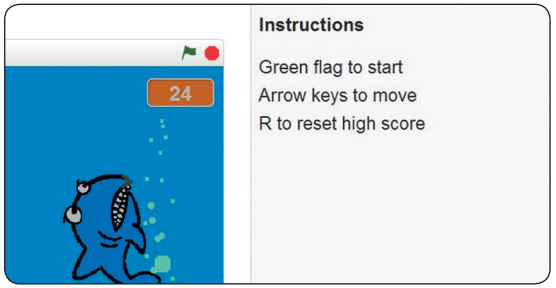
Step 6: Writing game instructions and publishing the game

RESOURCES

- Pupil self-assessment information

- Link to an example Scratch game, e.g. http://scratch.mit.edu/projects/15906446, http://scratch.mit.edu/projects/15906870 and http://scratch.mit.edu/projects/15907506

POSSIBLE OUTCOME FOR THIS STEP:



- If you have time, give the pupils a chance to test their game again, and make any final tweaks.
- Ask the pupils to think about any instructions or other information they need to include with their game, and how these might best be presented. Show the pupils the information included with the example Scratch games (see *Resources*).
- Suggest that the pupils include a splash screen at the beginning of their game and a short set of instructions, either as text or as recorded narration. Mention that giving too much information at this point might spoil the fun of the game for players.
- Show the pupils how games on the Scratch site can be remixed by others (i.e. other programmers make their own version of a Scratch project by changing the code, sprites, etc.). Encourage the pupils to add further comments to their scripts to explain how they work (by right clicking and choosing 'Add Comment'), so that if anyone does remix the game they'll be able to understand quickly how it works. Remind the pupils of the importance of using logical reasoning to explain this.
- If the pupils have Scratch accounts (which requires parental consent), they can upload their games to the Scratch website themselves; alternatively, you could upload the games to a shared class account (providing you have appropriate consent), or to the school learning platform or class blog for others to play.
- Encourage the pupils to review the feedback they receive on their games while keeping an eye on any inappropriate comments; this is easier to do on your school learning platform or class blog.
- Use a final plenary for the pupils to review their work on this project and discuss how their programming skills have developed.
- Finally, the children should evaluate the success of their work.

SCHOOL

- Some pupils could record a screencast in which they explain how their algorithms and scripts work, as well as discussing some of the difficulties they overcame in developing their game.

HOME

- Encourage the pupils to share their completed game with a wider circle of family and friends, feeding back on the reaction they received.

4 Assessment guidance

Use this page to assess the children's computing knowledge and skills. You may wish to use these statements in conjunction with the badges provided on the CD-ROM or community site and/or with your own school policy for assessing work.

ALL CHILDREN SHOULD BE ABLE TO:	BADGE	COMPUTING PoS REFERENCE
Create an algorithm for a game		Design programs that accomplish specific goals
Create images and sounds for use in their game		Create content
Use sequences of instructions		Use sequence in programs
Detect errors in their game		Use logical reasoning to detect errors in algorithms and programs
MOST CHILDREN WILL BE ABLE TO:		
Create music for use in their game		Create content
Use selection and repetition in their game		Use selection and repetition in programs
Correct errors in their game		Use logical reasoning to correct errors in algorithms and programs
Improve their game on the basis of the feedback they receive		Write and debug programs that accomplish specific goals
Add instructions to their game		Create programs, systems and content
SOME CHILDREN WILL BE ABLE TO:		
Break their game into its component parts and develop them separately		Solve problems by decomposing them into smaller parts
Create multiple images for characters and use them for animation		Create content
Use variables in their game		Work with variables
Explain how their game works		Use logical reasoning to explain how some simple algorithms work
Include comments in the code for their game		Use logical reasoning to explain how some simple algorithms work

PROGRESSION

The following units will allow your children to develop their knowledge and skills further.

- *Unit 6.4 – We are interface designers*
- *Unit 6.5 – We are mobile app developers*

5 Classroom ideas

Practical suggestions to bring this unit alive!

DISPLAYS AND ACTIVITIES

- Consider allowing the pupils to bring in commercial video games and games consoles for the first step of the unit, sharing their favourite games with others in the class.
- Encourage the pupils to work together to act out the instructions for the sprites in their game as a way to develop their understanding of their algorithms.
- The pupils' original artwork and written/drawn algorithms for their games can make an effective display, alongside screenshots from their completed game.
- When the games are complete, you could organise a tournament in the class or across the school.
- The games could be used on a stall at the school fête for fundraising.

WEBLINKS

- http://www.skillset.org/games has useful material on computer games, particularly in education.
- Some examples of game design projects: http://makethingsdostuff.co.uk/make-things/games.
- Code Club have extensive materials on programming projects for primary pupils, including some more structured material on game development, see www.codeclub.org.uk (registration required).
- Digital Schoolhouse has a Pac Man game activity at: www.resources.digitalschoolhouse.org.uk/key-stage-2-ages-7-10/142-scratch-lessons-free.

VISITS

- It would be worth pairing up with a younger class for this unit, getting your pupils to design and develop games for younger children. If so, a preliminary visit to the class would help the pupils pitch their games at an appropriate level.
- It would also be interesting to invite a game developer to discuss their work with the pupils, either in person or via video conference.

BOOKS

- Badger, M. *Scratch 1.4 Beginner's Guide*. (Packt Publishing, 2009)
- Burgun, K. *Game Design Theory: A New Philosophy for Understanding Games*. (CRC Press, 2012)
- Ford, J. *Scratch Programming for Teens*. (Delmar, 2008)
- Koster, R. *A Theory of Fun*. (O'Reilly Media, 2013)
- The LEAD Project. *Super Scratch Programming Adventure*. (No Starch Press, 2012)

6 Taking it further

When you've finished, you might want to extend the project in the following ways.

- Once the pupils have mastered programming in Scratch, they might like to explore the very different interface and tools available in Kodu.
- Encourage the pupils to think about the algorithms and programs that lie behind the computer games they play.
- The pupils can remix others' games on the Scratch community website.
- More advanced tools such as Game Salad (http://gamesalad.com) and GameMaker: Studio (www.yoyogames.com/studio) might appeal to talented or gifted pupils who have acquired expertise in Scratch.
- Text-based adventure games, known as 'interactive fiction', provide another approach to this unit. The Quest toolkit (http://textadventures.co.uk/quest) provides one set of tools for this, although it is possible to program games like this in a standard programming language such as Python.
- The pupils could apply the programming skills and knowledge they acquire in this unit to developing other programs, particularly games and interactive simulations, for one or more aspects of the curriculum. Science is a particularly rich source of ideas for this approach.