Unit 4.1: We are software developers
Developing a simple educational game

Software: Scratch (alternative: Snap!)
Hardware: Laptop/desktop/Chromebook computers or tablets, microphones (optional)

Overview

In this unit, pupils plan, create, develop and test their own educational game for a target audience. In:

- **Session 1** they analyse existing games and identify what makes them effective
- **Session 2** they create a working prototype of their game
- **Session 3** they develop the functionality of their game
- **Session 4** they improve the interface of their game
- **Session 5** they develop progression within their game
- **Session 6** they test and improve their game.

Alternatives

There are some alternatives to **Scratch**, such as Snap!, Blockly and Tynker. Some pupils may benefit from using ScratchJr rather than regular Scratch.

Knowledge, skills and concepts

In this unit, pupils will learn to:

- develop an educational computer game using selection and **repetition**
- understand and use **variables**
- start to **debug** computer **programs**
- recognise the importance of user interface design, including consideration of **input** and **output**.

Progression
In Key Stage 1:

- Pupils programmed using Blue-Bots in **Unit 1.1: We are treasure hunters**.
- They programmed on screen in **Unit 2.1: We are astronauts**.

In Key Stage 2:

- Pupils will build on learning from **Unit 3.1: We are programmers.**
- Pupils will further develop their programming skills in **Unit 4.2: We are makers** and **Unit 5.1: We are game developers.**

Assessment – by the end of the unit:

All pupils can:

- design an interactive educational game
- develop an interactive educational game
- put **Scratch** blocks into the right order for their game
- use the *if/then/else* block correctly
- use the keyboard for **input** and the screen for **output**.

Most pupils can:

- use a *repeat* block correctly
- keep track of random numbers and the score
- integrate sound into their game
- correct mistakes in their game.

Some pupils can:

- plan their own approach to developing their game
- use a countdown timer
- use the mouse to control the game
- explain how the **algorithm** that underlies their game works
- use logical reasoning to detect and correct **bugs** in their game.

Background information

Computer **programming** involves taking an **algorithm**, that is a **sequence** of instructions or a set of rules, and converting it into a language that a computer can follow. Here, the instructions are for a simple 'drill and practise' arithmetic game. These instructions have to be converted into the block-based programming language **Scratch**, where programs are built by snapping 'blocks' together for each instruction.

The programs here draw on the 'big six' ideas of coding: **sequence**, selection, **repetition**, **variables**, **input** and **output**. Any problem that can be solved by a computer can be solved using just these six ideas.

Key vocabulary

Algorithm: a sequence of precise instructions or steps (sometimes a set of rules) to achieve an objective

Bug: an error or mistake in a program or algorithm, causing the computer or robot to behave in a manner that was not originally intended

Debug: correct mistakes in a computer program or algorithm

Input: data supplied to a computer, in this case the algorithm taken from the storyboard for the animation

Output: information produced by a computer, in this case an animation

Program: a sequence of instructions (or sometimes a set of rules) that can be followed by a computer

Repeat loop: a sequence of instructions executed a fixed number of times or until some condition is met, or possibly forever

Repetition: programming construct which allows a group of instructions to be repeated a number of times, or until a certain condition is met

Scratch: simple, block-based programming language in which programs for characters are built by snapping together code blocks

Sequence: placing programming instructions in order, so each one happens one after the other

Sprite: a graphical character in a program that can be given its own sequence of instructions

Variable: lets computer programs store, retrieve or change simple data. Typically thought of as a particular location in the computer's memory that holds a specific item of data

Differentiation

See each session (pages 13–18) for ways to increase support and add challenge to this unit.

Although ScratchJr provides a greatly simplified interface, it cannot be used to create the programs described here. The unit can be made more accessible by providing pupils with partially completed programs, or the blocks needed and asking them to arrange these correctly, as if they were a jigsaw puzzle. Pupils learning EAL can change Scratch so that it works in their first language.

There are many ways in which pupils can be challenged in this unit, such as creating spelling test programs using speech synthesis or keeping track of which questions players found difficult.

Cross-curricular opportunities

Maths: Pupils can practise recall of multiplication and/or division facts, rounding decimals with one decimal place to the nearest whole number or converting between different units of measure in their games.

English: Pupils can create their own spelling tests using audio recording and playback.

Languages: Pupils can practise vocabulary in foreign languages in their games.

Other: Any subject where there are facts to learn can also provide a useful context, such as dates in history or capital cities in geography.

Preparation for teaching the unit

Things to do

- Decide which software/tools are most accessible/appropriate for use with your class. Although Scratch is recommended, Snap! is an alternative.
- Check you have access to the Scratch website and online editor. A downloadable version is also available (see *Useful links*).
- Spend some time familiarising yourself with your chosen software/tools.
- Work through the unit yourself so you know what is expected of pupils.
- Ensure you have sufficient computers/laptops/tablets and other equipment booked in advance.
- Look at some of the examples of educational games on the Scratch website.
- Decide which subject/topic the educational games will be linked to.

Resources needed

- **Software:** Scratch (see *Alternatives* page 10)
- **Hardware:** Laptop/desktop/Chromebook computers, microphones (optional)

Online resources provided

Session resources

- Worksheet 4.1a: Storyboard template
- Worksheet 4.1b: Prompt sheet
- Worksheet 4.1c: End-of-unit quiz
- Worksheet 4.1d: Pupil self-assessment
- Teaching slides 4.1a–4.1f
- Walkthrough videos 4.1a–4.1e
- Interactive end-of-unit quiz 4.1

Scratch projects, all saved in: scratch.mit.edu/studios/27279343/

- Addition race
- Spelling test
- 6 × 8 game:
- Multiplication game – adding variables
- Multiplication game – repetition
- Multiplication game – repetition (challenge)
- Multiplication game – interface
- Multiplication game – progression

Additional resources

- CPD video: Using selection
- CPD video: Working with variables
- Software in 60 seconds: Scratch 1–6b
- Software in 60 seconds: Scratch variables
- Software in 60 seconds: Scratch feedback

Online safety

- Pupils do not need accounts to download Scratch or to use Scratch or Snap! online.
- If pupils do register for accounts, they need to give a parent's or carer's email address. Check that parents/carers are happy about this.
- Once registered, pupils can share their work with the global Scratch community in a safe, moderated online space. Alternatively, pupils can upload games to a school platform.
- Pupils should respect licence conditions and intellectual property rights when incorporating images and sound effects from the Internet.

Collaboration

The planning here is based on pairs working collaboratively to discuss and decide the key features of good programming. They analyse how games work and suggest improvements, then work together to build an educational game.

Useful links

Software and tools

- Download Scratch: **scratch.mit.edu/download** or use online: **scratch.mit.edu/projects/editor**
- Snap! is free open-source software. Use online at: **snap.berkeley.edu/snapsource/snap.html**

Online tutorials

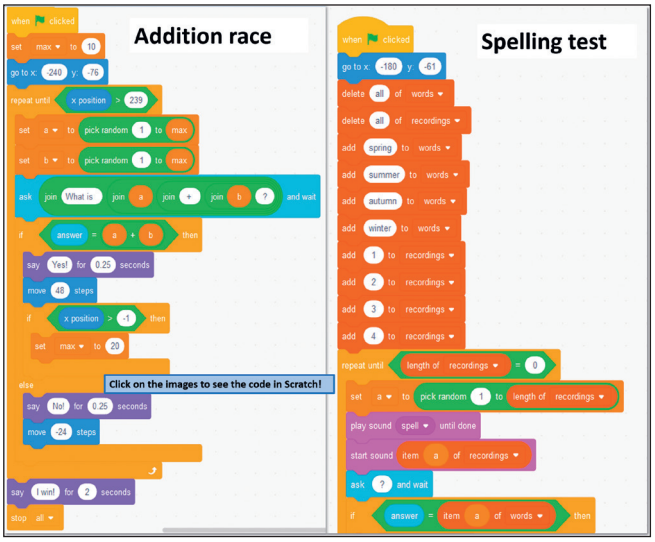
- Scratch tutorials built into editor
- Scratch Maths Quiz: **www.digitalschoolhouse.org.uk/workshops/the-maths-quiz**

Information and ideas

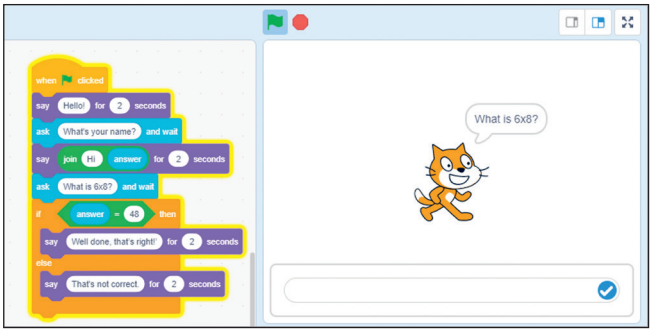
- Scratch resources for teachers: **scratch.mit.edu/educators/**
- Scratch educator community: **scratched.gse.harvard.edu**
- Snap! educators: **byob.berkeley.edu**
- Further ideas on selection: **www.sites.google.com/site/primaryictitt/home/key-stage-2/selection**; repetition: **www.sites.google.com/site/primaryictitt/home/key-stage-2/repetition**; and variables: **www.sites.google.com/site/primaryictitt/home/key-stage-2/variables**
- Educational games: **www.learninggamesforkids.com** **www.bbc.co.uk/bitesize/topics/zd2f7nb** **www.topmarks.co.uk/maths-games**

Unit outcomes

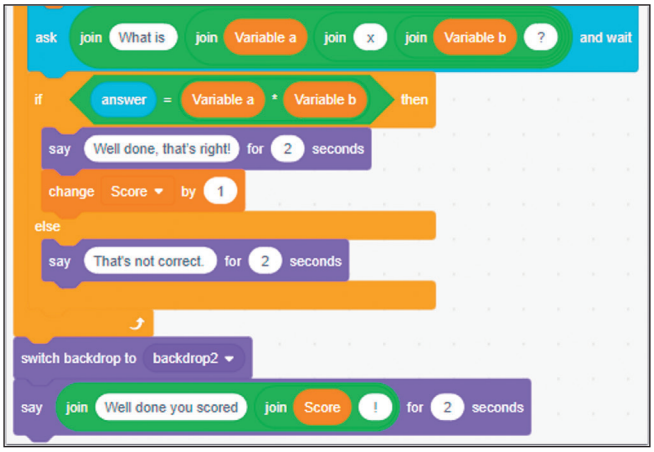
Below are some examples of the outcomes you could expect from this unit.



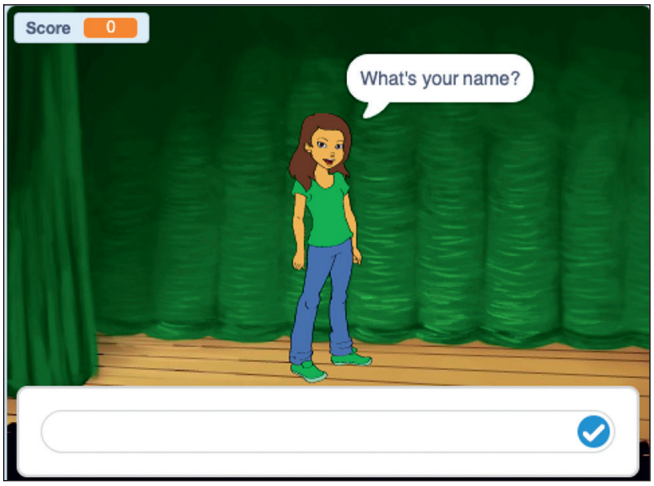
Session 1: Analysing an educational Scratch game



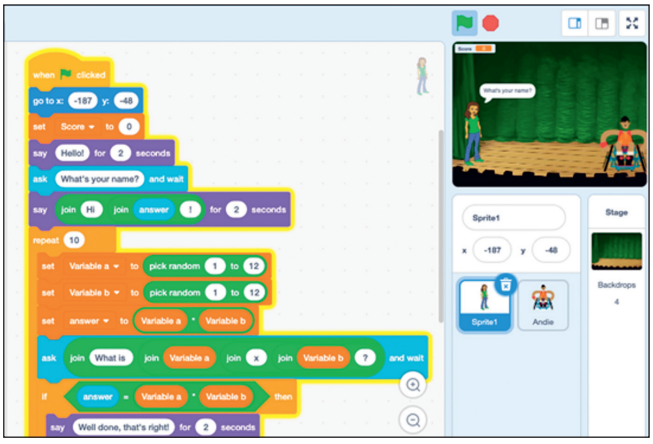
Session 2: Creating a simple program for a game



Session 3: Adding repetiton and messages



Session 4: mproving the interface of the game



Session 5: Building in progression

Session 6: Testing and refining the game