Unit 3.2: We are bug fixers

Finding and correcting bugs

Software: Scratch (alternative: Snap!), screen recorder software
Hardware: Laptop/desktop/Chromebook computers or tablets, microphones (if needed)

Overview

In this unit, pupils learn to recognise some common types of programming error, and practise solving problems through logical thinking. In:

- **Session 1** pupils identify and correct off-by-one **bugs**
- **Session 2** pupils identify and correct performance bugs
- **Session 3** pupils identify and correct multithread bugs
- **Session 4** pupils identify and correct conceptual bugs
- **Session 5** pupils identify and correct arithmetical bugs
- **Session 6** pupils identify and correct resource bugs.

Alternatives

Pupils could develop their **debugging** skills by fixing the **code** they and their classmates develop, for example by combining ideas from this unit with **Unit 3.1: We are programmers.** You do not have to use the example scripts provided. You could use your own scripts or your pupils' scripts, instead. Pupils could debug and/or develop programs downloaded from the **Scratch** website, which are all covered by a Creative Commons licence, i.e. they can be reused as long as the original author is credited and the resulting projects are shared on the same basis.

Knowledge, skills and concepts

In this unit, pupils will learn to:

- develop a number of strategies for finding errors in **programs**
- build up resilience and strategies for problem solving
- increase their knowledge and understanding of **Scratch**
- recognise a number of common types of **bugs** in software.

Progression

In Key Stage 1:

- Pupils fix sequences of instructions to get a Blue-Bot to a particular destination in **Unit 1.1: We are treasure hunters.**
- Pupils program a sprite on-screen in Scratch in **Unit 2:1: We are astronauts.**

In Key Stage 2:

- Pupils program in Scratch in **Unit 3:1: We are programmers.**
- Pupils will continue to develop Scratch programs and apply **debugging** skills to their own programs in **Unit 4.1: We are software developers** and **Unit 5.1: We are game developers.**

Assessment – by the end of the unit:

All pupils can:

- correct 'off-by-one' errors in loops
- improve the performance of the circle drawing **program**
- get the dialogue in the joke program to work in **sequence**
- experiment with the speed **variable** and other factors in the racing car simulator.

Most pupils can:

- describe how the times table program works
- describe how the circle drawing program works
- describe how the two joke scripts work together
- correct the 'Pong' style game so the bounce is more realistic
- describe how the racing car simulator works.

Some pupils can:

- explain how they **debugged** the times table program using logical reasoning
- explain the connection between the number of steps, step size and turn in the circle drawing program
- explain how they corrected the joke program
- describe how the 'Pong' style program works
- suggest explanations for the **bug** in the racing car simulator.

Background information

- Much of the work, and fun, in programming lies in spotting and correcting mistakes, known as '**bugs**'. The process of finding and fixing bugs is called '**debugging**'. In this unit, pupils will debug **programs** that accomplish specific goals.
- The more complex a program is, the more likely bugs are to occur. Debugging and developing others' projects is a great way for pupils to use **logical reasoning** to explain how simple **algorithms** work and to detect and correct errors in algorithms and programs.
- We call it debugging because when Admiral Grace Hopper was working on a Mark II computer at Harvard University in the 1940s, her associates discovered a moth stuck in a relay which stopped the Mark II from operating. Hopper remarked that they were 'debugging' the system.

Key vocabulary

Abstraction: computational thinking approach to managing complexity by simplifying things through identifying what is important, and what detail can be hidden or ignored

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

Code: instructions (or sometimes rules) that can be understood by a computer

Debug: correct mistakes in a computer program or algorithm

Event: something that happens within a computer program to cause some particular code to be run, such as an internal message being received, or a sprite being tapped by the user

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

Logical reasoning: to be able to give a reason for something which others would have to accept as correct

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

Parallel processing: when programs run (or appear to run) simultaneously

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

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: named storage location in a computer's memory

Differentiation

See each session (pages 23–28) for ways to increase support and add challenge to this unit.

For pupils learning EAL, show how they can **program Scratch** in their first language by selecting this from the globe icon at the top of the screen. This unit uses maths skills, so you may want to give extra support to pupils who struggle with maths. Some pupils may benefit from working with a partner, particularly for the final two steps.

Pupils can be challenged to modify or extend the example programs here, once they have successfully **debugged** them.

Cross-curricular opportunities

Art and design: Pupils discuss and critique the characters and backdrops in animations.

English: Pupils check the plot of the animation based on a traditional tale or picture book.

Maths: Explore times tables and shape drawing animations online.

Music: Pupils could suggest improvements to music for the animations/games.

Preparation for teaching the unit

Things to do

- Decide which software/tools are most accessible and appropriate for use with your class. Scratch is recommended and the example scripts provided are all built using Scratch (see *Resources needed*).
- Download your chosen software/tools (see *Useful links*), or ensure pupils have access to the Scratch website. They do not need to register for accounts.
- Read pages 20–21 to get an overview of the unit.
- Read the steps in the unit sessions (pages 23–28) and look at the associated online resources, printing out the worksheets as required.
- Watch the CPD videos.
- Work through the example scripts provided on the online resources and have a go at debugging them.
- Watch the walkthrough videos for this unit (see *Online resources*).
- Ensure you have enough computers/laptops/ tablets and other equipment booked in advance.
- Decide whether you need evidence of pupils' debugging – are corrected scripts sufficient, or do you want pupils to record screencasts? It is useful for pupils to be able to record an explanation of how they improved the scripts, but the time taken will detract from their programming.

Resources needed

- **Software:** Scratch (alternative: Snap!), screen recorder software
- **Hardware:** Laptop/desktop/Chromebook computers or tablets and microphones (if needed)

Online resources provided

Session resources

- Worksheet 3.2a–e: Debugging support sheets
- Worksheet 3.2f: End-of-unit quiz
- Worksheet 3.2g: Pupil self-assessment
- Teaching slides 3.2a–3.2f
- Walkthrough videos 3.2a–3.2g
- Six Scratch scripts (with bugs)
- Interactive end-of-unit quiz 3.2

Additional resources

- CPD video: Different approaches to debugging
- CPD video: Thinking about bugs
- CPD video: Screencasting
- CPD video: Multithreading

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, so you should check with parents or carers that they are happy for their children to do this.
- Once registered, pupils can share their corrected and refined programs with the global Scratch community in a safe online space. Alternatively, pupils can upload to a school platform.
- If pupils upload screencasts of their solutions, make sure you take the usual precautions to protect their identity.
- If pupils use the Internet for research, ensure all usual Internet safety protocols are in place.

Collaboration

Pairs work collaboratively to discuss the key features of a good game, suggest improvements and debug problems. Partners will work together to correct the programming of sprites and backdrops, so that the game is more interesting and it works more effectively.

Useful links

Software and tools

- Scratch: **scratch.mit.edu/download**
- Scratch Online: **scratch.mit.edu/projects/editor**
- Snap!: **snap.berkeley.edu/snapsource/snap.html**
- Screencast-o-matic: **www.screencast-o-matic.com/screen_recorder**

Online tutorials

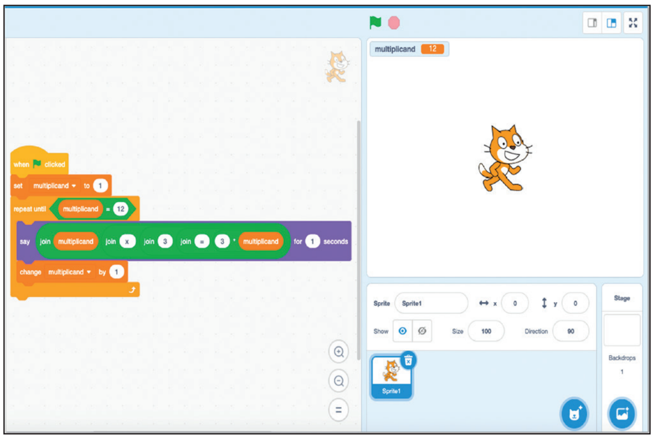
- Scratch tutorials are part of the editor
- Bug solutions: **youtu.be/grMMY2LSKFI**

Information and ideas

- Scratch project directory: **scratch.mit.edu/studios/27267652**
- Further debugging challenges: **scratch.mit.edu/studios/219583**
- Video clip 1: Creating a driving simulator: **www.bbc.co.uk/programmes/p016j4g5**
- Video clip 2: Simulating Formula 1 racing: **www.bbc.co.uk/programmes/p016612j**
- Video clip 3: Google's self-driving cars: **www.youtube.com/watch?v=cdgQpa1pUUE**

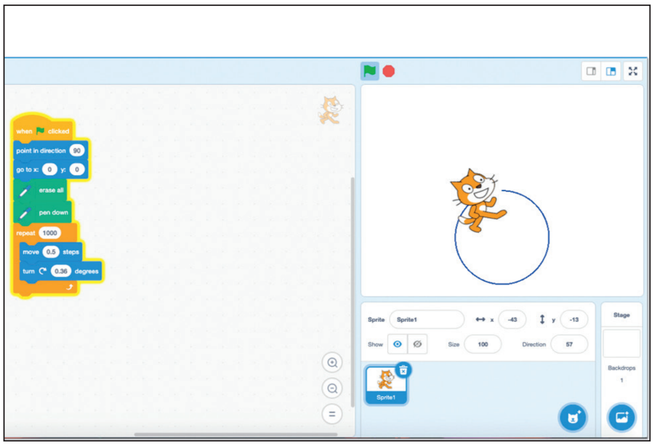
Unit outcomes

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



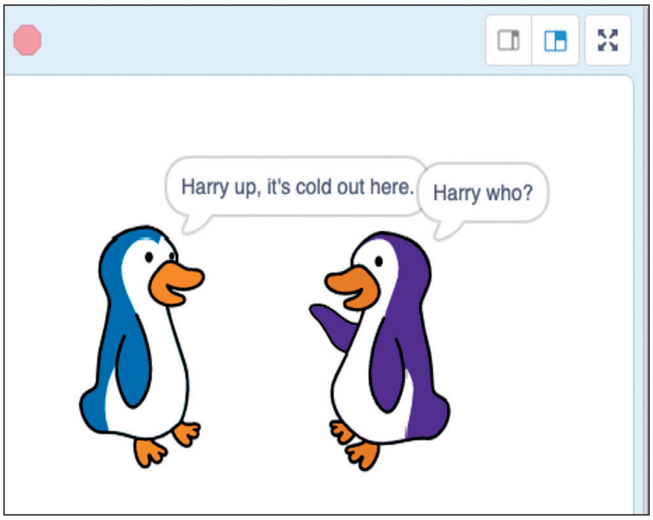
```
when ⚑ clicked
set multiplicand ▾ to 1
repeat until multiplicand = 12
    say join multiplicand join x join 3 join = 3 * multiplicand for 1 seconds
    change multiplicand ▾ by 1
```

Session 1: Fixing an off-by-one bug

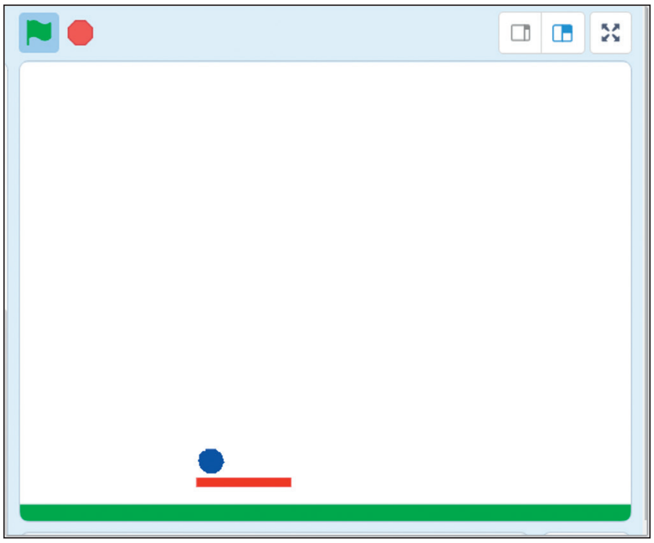


```
when ⚑ clicked
point in direction 90
go to x: 0 y: 0
erase all
pen down
repeat 1000
    move 0.5 steps
    turn ↻ 0.36 degrees
```

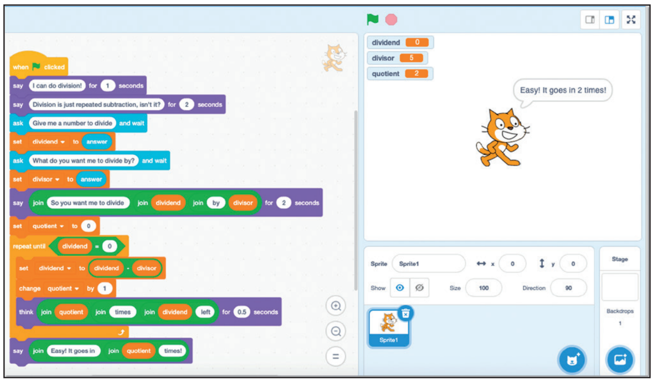
Session 2: Fixing a performance bug



Session 3: Fixing a multithread bug

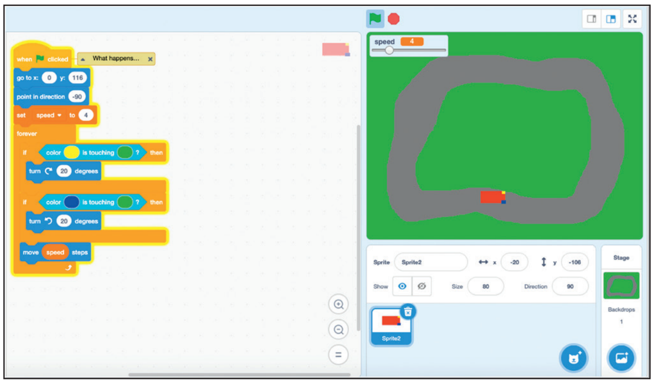


Session 4: Fixing a conceptual bug



```
when ⚑ clicked
say I can do division! for 1 seconds
say Division is just repeated subtraction, isn't it? for 2 seconds
ask Give me a number to divide and wait
set dividend ▾ to answer
ask What do you want me to divide by? and wait
set divisor ▾ to answer
say join So you want me to divide join dividend join by divisor for 2 seconds
set quotient ▾ to 0
repeat until dividend = 0
    set dividend ▾ to dividend - divisor
    change quotient ▾ by 1
    think join quotient join times join dividend left for 0.5 seconds
say join Easy! It goes in join quotient times!
```

Session 5: Fixing an arithmetical bug



```
when ⚑ clicked
go to x: 0 y: 116
point in direction -90
set speed ▾ to 4
forever
    if color 🟨 is touching 🟩 ? then
        turn ↻ 20 degrees
    if color 🟦 is touching 🟩 ? then
        turn ↺ 20 degrees
    move speed steps
```

Session 6: Fixing a resource bug