



Code Kingdoms

Unit 1

Introducing Code Kingdoms



for kids, with kids, by kids.

Resources overview

We have produced a number of resources designed to help people use Code Kingdoms. There are introductory guides to all parts of the product and classroom materials to help teach lessons around Code Kingdoms.



Code Kingdoms Learning: What, where, when and how

A summary of the Code Kingdoms approach to learning.



Teacher Guide

An overview for teachers. Describes the Code Kingdoms learning ethos and details the different parts of the product.



Dashboard Guide

A beginner's guide to using our group management tool. Describes everything from registering for an account to assessing the progress of your kids.



Sandbox guide

A guide to using our unstructured creation environment. Learn everything from using the menus to making great puzzles.



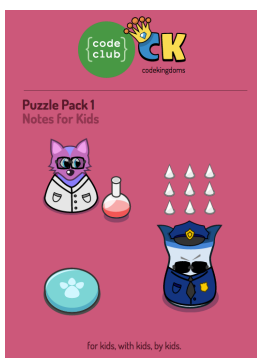
Unit 1: Introducing Code Kingdoms

An introductory unit of six 'off-the-shelf' lesson plans. Targeted at KS2 kids.



Unit 2: Learning a language

Six 'off-the-shelf' lesson plans designed to teach kids the basic of JavaScript

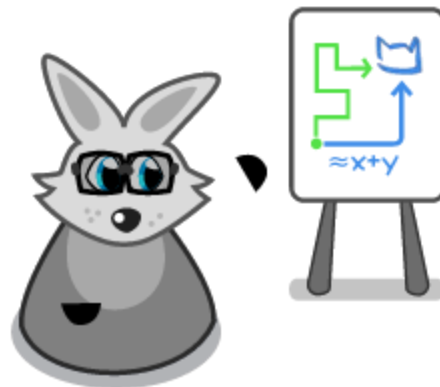


Puzzle Packs

A guide to building specific puzzles in Creative mode. Step-by-step instructions from start to finish. Four puzzles per pack.

Contents

About this guide	2
Unit Plan	3
Lesson 1	6
Lesson 2	9
Lesson 3	13
Lesson 4	17
Lesson 5	21
Lesson 6	21
Map Creation Checklist	26
Map Creation Assessment Sheet	27



About this guide

This scheme of work contains six one hour lesson plans, a medium term plan for the unit and some accompanying assessment grids. It is designed as an introduction to Code Kingdoms and can be used across the age ranges. The lesson plans are meant to be flexible, rather than prescriptive, to allow teachers to adapt it as they see fit.

This unit develops skills in three strands of learning: JavaScript, Computational Thinking & Computing. The learning outcomes for each lesson are categorised in these three areas. The unit culminates in the kids building their own map which will allow them to demonstrate their learning throughout the unit.



Resources



To teach this unit you will require the following resources:



- CK School - school.codekingdoms.com
- CK Dashboard - dashboard.codekingdoms.com

You may find our supporting guides useful. They are found at codekingdoms.com/teachers

- Dashboard guide - how to use the teacher management tool to plan and set activities
- Sandbox mode guide - how to use the creation environment to complete lesson plan activities

Unit Plan


	Lesson Title	Outcomes
	Commands, cause and effect and parameters	<div data-bbox="683 464 776 527">1.1</div> I understand how to execute basic JavaScript. <div data-bbox="683 541 776 604">1.2</div> I understand how parameters can affect the output of a function. <div data-bbox="683 657 776 720">PA2</div> I know that computers need precise instructions. <div data-bbox="683 735 776 798">PP3</div> I can run, check and change programs.
	Spikes and buttons	<div data-bbox="683 877 776 940">1.2</div> I understand how parameters can affect the output of a function. <div data-bbox="683 993 776 1056">1.3</div> I understand how to use JavaScript functions to achieve my aims. <div data-bbox="683 1150 776 1213">A1</div> Writing instructions that if followed in a given order (sequences) achieve a desired effect. <div data-bbox="683 1266 776 1329">E1</div> Assessing that an algorithm is fit for purpose; <div data-bbox="683 1423 776 1486">YA4</div> I can find and correct errors i.e. debugging, in algorithms. <div data-bbox="683 1501 776 1564">YP2</div> I can use logical reasoning to predict the behaviour of programs.

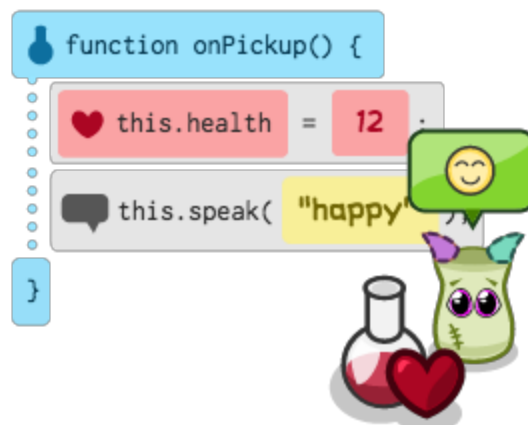
	Lesson Title	Outcomes
	Glitch Behaviour 1	1.3 I understand how to use JavaScript functions to achieve my aims.
		A1 Writing instructions that if followed in a given order (sequences) achieve a desired effect.
		E1 Assessing that an algorithm is fit for purpose;
		YA4 I can find and correct errors i.e. debugging, in algorithms.
		YP2 I can use logical reasoning to predict the behaviour of programs.
	Glitch Behaviour 2	1.2 I understand how parameters can affect the output of a function.
		2.4 I can explain how to use a while loop to solve a practical problem.
		A5 Writing instructions that repeat groups of constituent instructions (loops/iteration) to achieve a desired effect;
		D1 Breaking down artefacts into constituent parts to make them easier to work with;
		YA1 I know that algorithms are implemented on digital devices as programs.
		YA2 I can design simple algorithms using loops, and selection i.e. if statements.
		YA3 I can use logical reasoning to predict outcomes.
		YP1 I can use arithmetic operators, if statements, and loops, within programs.

Lesson Title	Outcomes
<div>5</div> Creating your own map	<div>1.3</div> I understand how to use JavaScript functions to achieve my aims.
	<div>2.4</div> I can explain how to use a while loop to solve a practical problem.
	<div>A1</div> Writing instructions that if followed in a given order (sequences) achieve a desired effect.
	<div>A5</div> Writing instructions that repeat groups of constituent instructions (loops/iteration) to achieve a desired effect;
<div>6</div>	<div>D1</div> Breaking down artefacts into constituent parts to make them easier to work with;
	<div>E1</div> Assessing that an algorithm is fit for purpose;
	<div>YA1</div> I know that algorithms are implemented on digital devices as programs.
	<div>YA2</div> I can design simple algorithms using loops, and selection i.e. if statements.
	<div>YA3</div> I can use logical reasoning to predict outcomes.
	<div>YA4</div> I can find and correct errors i.e. debugging, in algorithms.
	<div>YP2</div> I can use logical reasoning to predict the behaviour of programs.

Lesson One

Commands, Cause and Effect and Parameters

Lesson Title	Outcomes
 Commands, cause and effect and parameters	1.1 I understand how to execute basic JavaScript.
	1.2 I understand how parameters can affect the output of a function.
	PA2 I know that computers need precise instructions.
	PP3 I can run, check and change programs.




Lesson Activities

Activity	Notes
<p>Introduction <i>(7-10 mins)</i></p> <ol style="list-style-type: none"> 1. Discussion incorporating key questions: <ul style="list-style-type: none"> ○ What is Coding/Programming? ○ What things use Coding? ○ Why is Coding useful? 2. Ask the kids to think about what a command is and where they are used. Lead them to real-world examples from the classroom (“pens down” “not talking”) and common commands given to dogs. 	<ul style="list-style-type: none"> ○ Code is a language that a machine understands to allow it to perform tasks that it was designed for. We use commands to tell these machines what to do. Different machines understand different languages. A food recipe is a piece of code that a human can understand and use to make lunch! ○ Example devices include mobile phones, manufacturing robots, cars, alarm clocks, rockets. Uses include making games, the internet and online shopping, cutting the carpet tiles the right size for the classroom.
<p>Using Commands <i>(15 mins)</i></p> <ol style="list-style-type: none"> 1. Kids login to Code Kingdoms and complete two Adventure packs from the Dashboard. This should familiarise kids with the format of the problem-solving aspect of Code Kingdoms. 2. Ask kids to identify any commands they come across/use. 3. Pool the list of commands kids encountered in when playing Adventures. 	<ul style="list-style-type: none"> ○ Ensure kids are aware where the code features in interactions with characters.

Activity	Notes
<p>Introduce Cause and Effect <i>(15 mins)</i></p> <ol style="list-style-type: none"> 1. Explore what cause and effect is using real-world examples [if you drop a pencil it will fall, press a light switch light will turn on] 2. Set two more Adventure packs and ask kids to spot at least two cause and effects in the world. 3. Pool causes and effects encountered in the game and ask kids to come up with a cause and effect they have experienced in the real-world today. 	<ul style="list-style-type: none"> ○ Provide kids with an example of cause and effect in the game. e.g. Codeling stands on a button causing a bridge to raise. ○ Number of Adventure packs set can be adjusted according to time.
<p>Introduce Parameters <i>(15 mins)</i></p> <ol style="list-style-type: none"> 1. How did the Codelings move? Direction? Speed? Ask kids to think about why these might be different. 2. Kids list the four compass directions that are used as parameters in the game. 3. Play the next map and identify any parameters they come across. Give them examples such as speed and direction that are given to Codelings. 	
<p>Plenary <i>(5 mins)</i></p> <ol style="list-style-type: none"> 1. In pairs kids act as a robot and a coder and carry out the following: <ul style="list-style-type: none"> ○ give a command (e.g. talk, jump etc.) ○ a cause and effect (when I clap, you jump) ○ a parameter (walk in a certain direction) 2. Switch roles and repeat. 	<ul style="list-style-type: none"> ○ Alternative: teacher/chosen kid calls out a mixture of commands, cause and effect and parameters for the whole class. kids act out instruction and call out whether they think it involves a command, cause and effect or parameter.

Lesson Two

Spikes and Buttons

	Lesson Title	Outcomes
	Spikes and buttons	1.2 I understand how parameters can affect the output of a function.
		1.3 I understand how to use JavaScript functions to achieve my aims.
		A1 Writing instructions that if followed in a given order (sequences) achieve a desired effect.
		E1 Assessing that an algorithm is fit for purpose;
		YA4 I can find and correct errors i.e. debugging, in algorithms.
		YP2 I can use logical reasoning to predict the behaviour of programs.

Lesson Activities

Activity	Notes
<p>Introduction <i>(7-10 mins)</i></p> <ol style="list-style-type: none"> 1. Identify sequencing as putting things in the correct order. Ask kids to think of criteria for sequencing class members and then to sequence themselves according to their criteria (e.g. height order, birthdays etc.) 2. Ask kids why sequencing programming instructions might be important. To guide their thinking as them to think what would happen if the following were to be sequenced incorrectly: <ul style="list-style-type: none"> ○ Open door ○ Walk through doorway ○ Close door 	<ul style="list-style-type: none"> ○ One kid could call out the commands in the wrong order and ask another kid to follow the commands. ○ Crossing the road is a suitable alternative to walking through a doorway.
<p>Kids play introductory Adventures <i>(10 mins)</i></p> <ol style="list-style-type: none"> 1. Kids login to Code Kingdoms and complete two Adventure packs to remind themselves of the game functionality and computational thinking required. 2. Tell kids they will be creating their own maps and should think about what they might do if they had to create a map. 	<ul style="list-style-type: none"> ○ Ensure kids are aware where the code features in interactions with characters.


Activity	Notes
<p>Introduce OOSY Method in Sandbox <i>(15 mins)</i></p> <ol style="list-style-type: none"> Set Sandbox mode from the Dashboard and ask kids to work out the aim of the map they are looking at. <ul style="list-style-type: none"> Objective -release the animal from the cage - point out the cage in the map. Next lead them through the framework for creating meaningful maps using examples. <ul style="list-style-type: none"> Obstacle - place spikes to block access to the bridge - impossible to complete! Solution - place button and add code to it. Your Turn! - give the map to someone else to test it. Kids replicate the obstacle (spikes) and solution (button) on their own computers. 	<ul style="list-style-type: none"> Display OOSY Method clearly in the classroom as it will be referred to extensively as kids progress through the unit.
<p>The 3Ws for structuring algorithms <i>(15mins)</i></p> <ol style="list-style-type: none"> Using the initial button and spikes example, help kids structure their commands: <ul style="list-style-type: none"> When? the button is pressed onPress Who? SpikesA What? lower() Add complexity to the spike obstacle: <ul style="list-style-type: none"> Select button onRelease Choose Spikes and drag in SpikesA.raise(); Try playing it - impossible! Ask kids to come up with a solution to solve the puzzle on their own island templates. 	<ul style="list-style-type: none"> Display the 3Ws in the classroom alongside the OOSY Method as it will help kids structure their algorithms. Depending on ability/experience kids may need to be guided to placing the crate to keep the button pressed down.

Activity	Notes
<p>Plenary <i>(10mins)</i></p> <ol style="list-style-type: none"> Kids use the 3Ws to describe some character behaviour they would like to see in next lesson in a sentence. E.g. When button is pressed the cat jumps. Peer assess each others sentence checking for algorithm structure and spelling, punctuation and grammar. 	<ul style="list-style-type: none"> ○ Retain kids' 3Ws sentences in preparation for next lesson.



Lesson Three

Glitch Behaviour 1

	Lesson Title	Outcomes
	Glitch Behaviour 1	<div data-bbox="630 558 727 625">1.3</div> I understand how to use JavaScript functions to achieve my aims. <div data-bbox="630 674 727 741">A1</div> Writing instructions that if followed in a given order (sequences) achieve a desired effect. <div data-bbox="630 789 727 856">E1</div> Assessing that an algorithm is fit for purpose; <div data-bbox="630 905 727 972">YA4</div> I can find and correct errors i.e. debugging, in algorithms. <div data-bbox="630 1020 727 1087">YP2</div> I can use logical reasoning to predict the behaviour of programs.

Cross-curricular links: Development of literacy skills in plenary activity. Calculating speed for Glitch movement.

Lesson Activities

Activity	Notes
<p>Introduction <i>(5 mins)</i></p> <ol style="list-style-type: none"> 1. Recap the OOSY Method, what it stands for and how it is used when creating maps. 2. Kids review their sentences from the previous lesson's plenary to remind them how to use the 3Ws when structuring an instruction. 3. Tell kids they will be coding character behaviour and that the 3Ws will be important. 	
<p>Create simple obstacle & solution <i>(10 mins)</i></p> <ol style="list-style-type: none"> 1. Load the Sandbox mode and place 3 Glitches in the map as an obstacle to reaching the cage. 2. Ask kids to copy the demonstrated template and to come up with a simple solution to get past the Glitches on their own computers. 	


Activity	Notes
<p>Kids code Glitch behaviour <i>(15 mins)</i></p> <ol style="list-style-type: none"> Using the 3Ws demonstrate coding some basic Glitch behaviour. E.g. <ul style="list-style-type: none"> ○ When? onCreate ○ Who? GlitchA ○ What? GlitchA.walkTowards(player); Kids give some basic behaviours to their own 3 Glitches. These might include - walk(direction), walkTowards(object) and jump Kids will likely find their Glitches are moving too slowly so they can adjust their speed. Speed is measured in metres per second (roughly one square per second) - kids can experiment with different speeds for their Glitches. 	<ul style="list-style-type: none"> ○ Opportunity for kids to differentiate for themselves based on the difficulty of behaviour they select. ○ Speed can be adjusted when the Glitch is selected in Editor mode. Speed is listed under the Motion heading in the Sequencer.
<p>Kids code more complex behaviours <i>(15 mins)</i></p> <ol style="list-style-type: none"> Adding to the existing sequence for Glitch behaviour, kids code a Glitch to return it to its starting position. Kids can then add further sequenced commands to their Glitches at their own pace. At this point kids may want to refer to their sentences from last lesson's plenary to see if they are able to code the behaviour they described. 	<ul style="list-style-type: none"> ○ Kids encouraged to plan the order they want the Glitch behaviours to be seen so that they are sequenced correctly in the code. ○ Remind kids to stick to the 3Ws even when they have freedom to decide on their Glitch behaviour.

Activity	Notes
<p>Plenary <i>(10 mins)</i></p> <ol style="list-style-type: none"> Kids write at least 4 correctly-sequenced instructions they would like to give to Glitches/Animals in the next lesson. Peer assess each others sentences checking for sequencing spelling, punctuation and grammar. 	<ul style="list-style-type: none"> ○ Written in full sentences rather than code. ○ Retain sentences for subsequent lesson.



Lesson Four

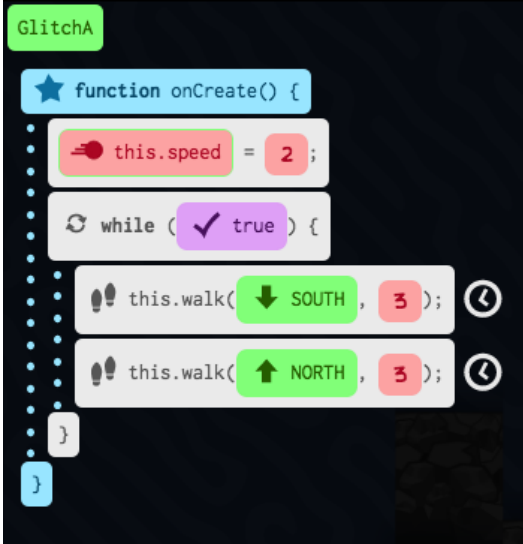
Glitch Behaviour 2

	Lesson Title	Outcomes
	Glitch Behaviour 2	1.2 I understand how parameters can affect the output of a function.
		2.4 I can explain how to use a while loop to solve a practical problem.
		A5 Writing instructions that repeat groups of constituent instructions (loops/iteration) to achieve a desired effect;
		D1 Breaking down artefacts into constituent parts to make them easier to work with;
		YA1 I know that algorithms are implemented on digital devices as programs.
		YA2 I can design simple algorithms using loops, and selection i.e. if statements.
		YA3 I can use logical reasoning to predict outcomes.
		YP1 I can use arithmetic operators, if statements, and loops, within programs.

Cross-curricular links: Development of literacy skills in plenary activity. Calculating speed for Glitch movement.

Lesson Activities

Activity	Notes
<p>Introduction <i>(5-7 mins)</i></p> <ol style="list-style-type: none"> 1. Kids review previous lesson's plenary sentences to remind themselves of sequencing commands. 2. When introducing the next task explore through questioning what the term 'efficiency' means and how it could help when coding. 	
<p>Patrolling Glitches <i>(12-15mins)</i></p> <ol style="list-style-type: none"> 1. Load Sandbox mode and ask kids to think about what might be a good obstacle for this map - lead them to the idea of patrolling Glitches. 2. Next ask kids to suggest a suitable place for the Glitch to patrol so that it is an obstacle to completing the map but not impossible. 3. How many code commands do we need for a Glitch to patrol? What are they? 4. Kids give the Glitch those commands on their own computers and asked to spot what is wrong with the code commands as they are. 5. Kids easily adjust speed parameters as before so that the Glitch is quick enough but the player can still pass to reach the cage. 6. How could we repeat the behaviour more times? What is an efficient way to code this? 	<ul style="list-style-type: none"> ○ 2 commands: <ul style="list-style-type: none"> ● GlitchA.walk(SOUTH);and ● GlitchA.walk(NORTH); ○ The problem with the code is the Glitch moves too slowly and stops after one patrol. ○ Kids will likely suggest repeating the instructions again and again but link back to earlier discussions about efficiency.

Activity	Notes
<p>Kids create while loops for efficient code (20 mins)</p> <ol style="list-style-type: none"> 1. Introduce while loops (also called forever loops) using real-world examples - we forever loop through the same 7 days of the week, the moon always orbits the Earth, a heart is always beating (during life). 2. In the Sequencer, demonstrate how a while loop is used with the existing north / south commands to make the Glitch patrol continuously (shown on the right). 3. Kids create their own while loop for patrolling Glitches in the same location as demonstrated. 4. Then create their own while loop for a second Glitch in a different location. 	<p>○ While loops are found under the 'Language' tab of the Sequencer.</p> 
<p>Using spikes & buttons with patrolling Glitches (12 mins)</p> <ol style="list-style-type: none"> 1. In some locations it will be impossible for the player to pass a patrolling Glitch. Ask kids to think about the methods they have used in previous lessons to provide solutions to obstacles. 2. Tell the kids they will be creating their own map in the next lessons and they should be as creative with their solution as possible. 	<p>○ The most straightforward solution that kids can be guided to is luring a Glitch onto some raised spikes.</p>

Activity	Notes
<p>Plenary <i>(5mins)</i></p> <ol style="list-style-type: none"> Kids each describe 3 real-world examples where they think while loops are used or would be useful. 	



Lessons Five and Six

Creating your own map

This section can be delivered as a double period or two distinct lessons, as such prescribed timings are flexible. It consolidates kids' learning from the unit allowing them to display their learning by creating their own map..

Lesson Title	Outcomes
<div data-bbox="159 688 230 760">5</div> Creating your own map	<div data-bbox="613 688 701 760">1.3</div> I understand how to use JavaScript functions to achieve my aims. <div data-bbox="613 802 701 873">2.4</div> I can explain how to use a while loop to solve a practical problem. <div data-bbox="613 915 701 987">A1</div> Writing instructions that if followed in a given order (sequences) achieve a desired effect. <div data-bbox="613 1029 701 1100">A5</div> Writing instructions that repeat groups of constituent instructions (loops/iteration) to achieve a desired effect;
<div data-bbox="159 1192 230 1264">6</div>	<div data-bbox="613 1192 701 1264">D1</div> Breaking down artefacts into constituent parts to make them easier to work with; <div data-bbox="613 1306 701 1377">E1</div> Assessing that an algorithm is fit for purpose; <div data-bbox="613 1419 701 1491">YA1</div> I know that algorithms are implemented on digital devices as programs. <div data-bbox="613 1533 701 1604">YA2</div> I can design simple algorithms using loops, and selection i.e. if statements. <div data-bbox="613 1646 701 1717">YA3</div> I can use logical reasoning to predict outcomes. <div data-bbox="613 1759 701 1831">YA4</div> I can find and correct errors i.e. debugging, in algorithms. <div data-bbox="613 1873 701 1944">YP2</div> I can use logical reasoning to predict the behaviour of programs.

Cross-curricular links: Survey design. Time management. Self-assessment in plenary.

Lesson Activities

Activity	Notes
<p>Introduction <i>(7 mins)</i></p> <ol style="list-style-type: none"> 1. In preparation for building their own map, ask kids to think about their favourite Code Kingdoms environment they have played. They should list what they liked about it most and if there was anything they would change. 	
<p>Planning / Design <i>(15-20 mins)</i></p> <ol style="list-style-type: none"> 1. Share the checklist / assessment criteria with kids. 2. Remind them of the OOSY Method and ask them to plan their map objective and at least 3 obstacles with solutions. 3. Kids should be encouraged to incorporate aesthetic design by choosing a theme for their map and deciding on some decorative items. 	<p>○ E.g. coastal, mountain, tundra, grassland, glitchy, forest. Depending on the kids' capabilities they can be allowed to create their map from the existing Sandbox template rather than designing from scratch.</p>

Activity	Notes
<p>Implementation <i>(30 mins)</i></p> <ol style="list-style-type: none"> Kids build their map making reference to OOSY, 3Ws and assessment criteria. Encourage kids to plan their time as well as their map structure so they don't spend too long decorating their map or adding too many coins or potions. 	
<p>Survey design <i>(15 mins)</i></p> <ol style="list-style-type: none"> Kids design a survey for people playing their map. Minimum key questions to ask testers: <ul style="list-style-type: none"> <input type="radio"/> Was the map fun? <input type="radio"/> Could you complete the map? <input type="radio"/> Was it too easy/too difficult/just right? <input type="radio"/> What could make the map better? Tell kids they will be creating their own map in the next lessons and they should be as creative with their solution as possible. 	<ul style="list-style-type: none"> <input type="radio"/> This stage can be given greater / less emphasis depending on how survey design fits into your kids' current programme of study.
<p>Testing/Debugging <i>(10 mins)</i></p> <ol style="list-style-type: none"> Once kids have coded at least 2 obstacles in their map they should give it to a partner to play and test its effectiveness. Testers should fill in the designers survey after playing. 	

Activity	Notes
<p>Improvements and Analysis (Plenary) <i>(20-30 mins)</i></p> <ol style="list-style-type: none"> 1. Kids finish building their map taking into account the feedback from the survey. 2. Kids complete a self-assessment of their map against the assessment criteria. 	

Map Creation Checklist

Tick off the following when you have included each one in your plan and your completed map design.
The peer review column is for the person checking your map.

	In your plan?	In your map?	Peer review
You have chosen your terrain			
There is a cage to end the map			
You have an obstacle that uses one line of code <input type="radio"/> E.g. spikes lower when a button is pressed			
You have an obstacle that uses more than one line of code <input type="radio"/> E.g. spikes lower when a button is pressed and raised when depressed			
You have an obstacle that uses a while loop <input type="radio"/> E.g. a patrolling Glitch that doesn't stop			
You have sequenced your code commands correctly			
All your obstacles have solutions <input type="radio"/> E.g. Glitches can be chased away			
You have decorated your map using trees, shrubs, etc.			
Your map can be completed by a player			

What do you think are the best parts of your map?

What do you think could be improved?

What was the most complicated part of your map?

Describe 3 great things about this map.

Describe 3 things that could make this map even better.

Map Creation Assessment Sheet

Student:	Red	Amber	Green
has designed their terrain well			
has placed a cage at the end of their map (Objective)			
has coded an obstacle using a simple algorithm ○ E.g. spikes lower when a button is pressed			
has coded an obstacle using a more complex algorithm ○ E.g. spikes lower when a button is pressed and raised when depressed			
has coded behaviour using a while loop ○ E.g. a patrolling Glitch that doesn't stop			
has correctly sequenced commands in an algorithm			
has provided solutions to ALL their obstacles			
has considered the aesthetics of their map			
has created a map that can be played and completed			

The really great things about your map are:

Next time you need to improve:

Planning Your Own Map

You can use the OOSY Method poster to help you design your map.

What is the objective of your game?

What objects (characters) will there be in your game and what will they do?

E.g. Giant Glitch – chases player around the map and aims to destroy it.

What obstacles will there be in your game?

What solutions will there be for the obstacles?



Planning Algorithms

When you create your map, the different objects within your game will need instructions to tell them what to do. Use this worksheet to help you consider what those instructions might be. An example has already been done for you.

The OOSY Method

When faced with a blank template and the task of creating a Code Kingdoms map many students will find it difficult to create a playable map.

The OOSY Method will assist students' planning by scaffolding the method of building Code Kingdoms maps.

OBJECTIVE	First a student must think of an objective for their map. For simplicity in our maps the player always has to reach the rocket.
OBSTACLE	Now the pupil should put an obstacle in the way of reaching the objective that makes the map impossible to complete. E.g. placing spikes on a bridge as an impasse.
SOLUTION	Now think of a solution that will let the player get round the obstacle. E.g. if the player places a lot of Glitches then they could now place a net which the player can pick up and catch them with.
YOUR TURN!	Share the map with friends to see if they can complete it!

The 3Ws

To assist students in thinking about structuring lines of code, we recommend using the 3Ws (When? Who? What?)

When?	onCreate
Who?	Glitch A
What?	walk towards player

This will allow students to create an algorithm where a Glitch will walk towards a player when it is created, which will look like:

```
GlitchA.walkTowards(player);
```

This method help students to sequence their commands logically.