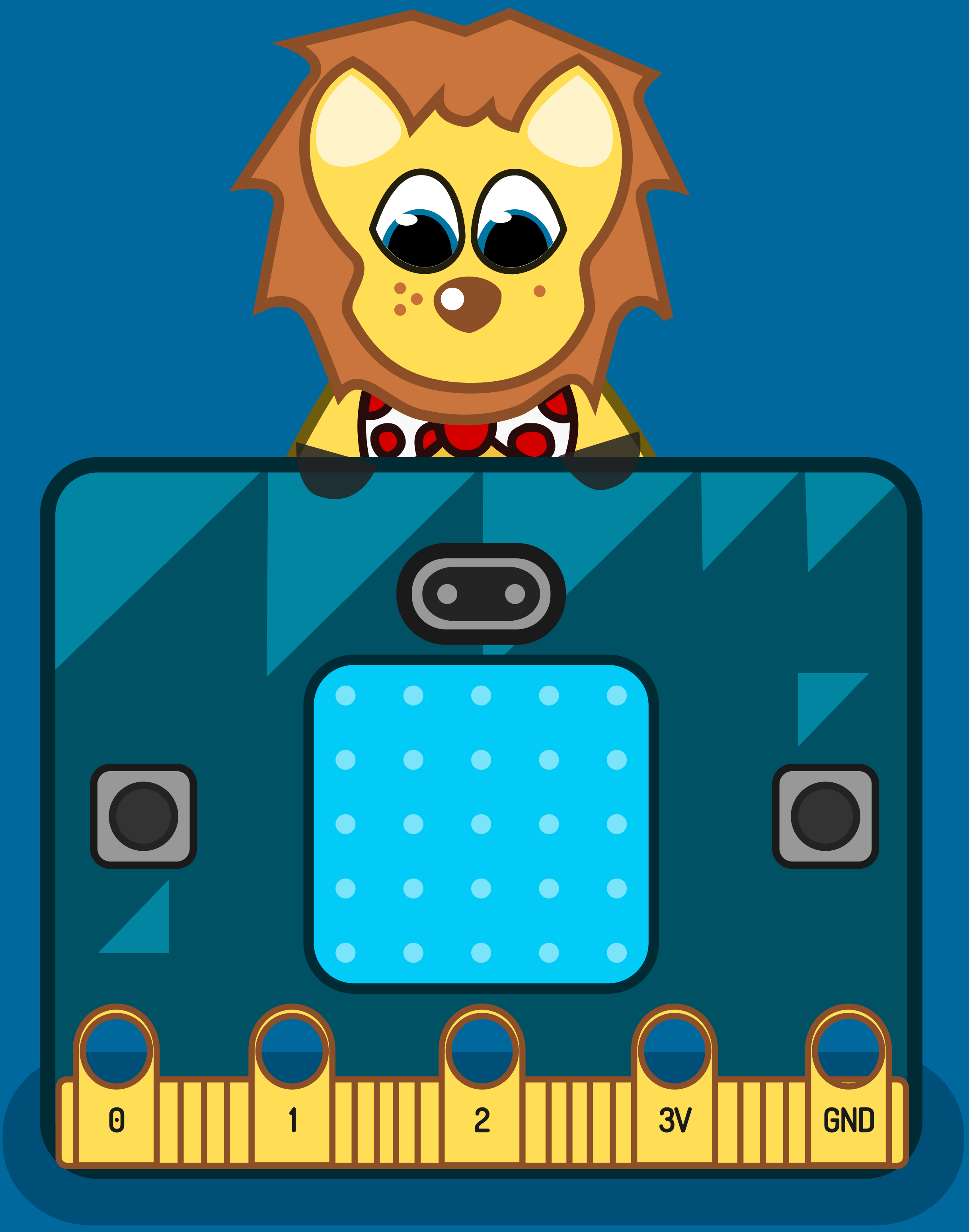
BBC micro:bit Cookbook

Using the Code Kingdoms JavaScript editor



for kids, with kids, by kids.

Code Kingdoms Cookbook

Contents

Introduction

The Code Kingdoms JavaScript editor that we are contributing to the BBC micro:bit began life as part of **Code Kingdoms**, a game that teaches children to code. We have decoupled and improved the editor to sit alongside the Python editor, Block Editor and Touch Develop as one of the tools to program the BBC micro:bit.

Our drag-and-drop interface makes the editor accessible to beginners and our ‘slider’ supports the transition to text-based programming as learners’ coding skills progress.

This booklet guides you through six different games of increasing difficulty to program your BBC micro:bit.

The first three games have step-by-step instructions, or recipes, that make familiarisation with different parts of the editor straightforward. The latter three games require you to build upon the skills and knowledge learned in the early games and, therefore, provide less guidance. Each activity has an idea for an extension which will add complexity to the game.

All of the recipes are supported by the tutorial system within the editor. Users can choose the game they want to build from the tutorial tab in the left sidebar, and the editor will guide them towards completion. As with this written guide, the first three tutorials provide much more support and are primarily aimed at being used with the visual editor, whereas the latter three will be based in the textual view of the editor and provide a skeleton of the code that needs to be written.

The coded solutions to each game are listed on the **Challenges and Solutions** page of this guide.

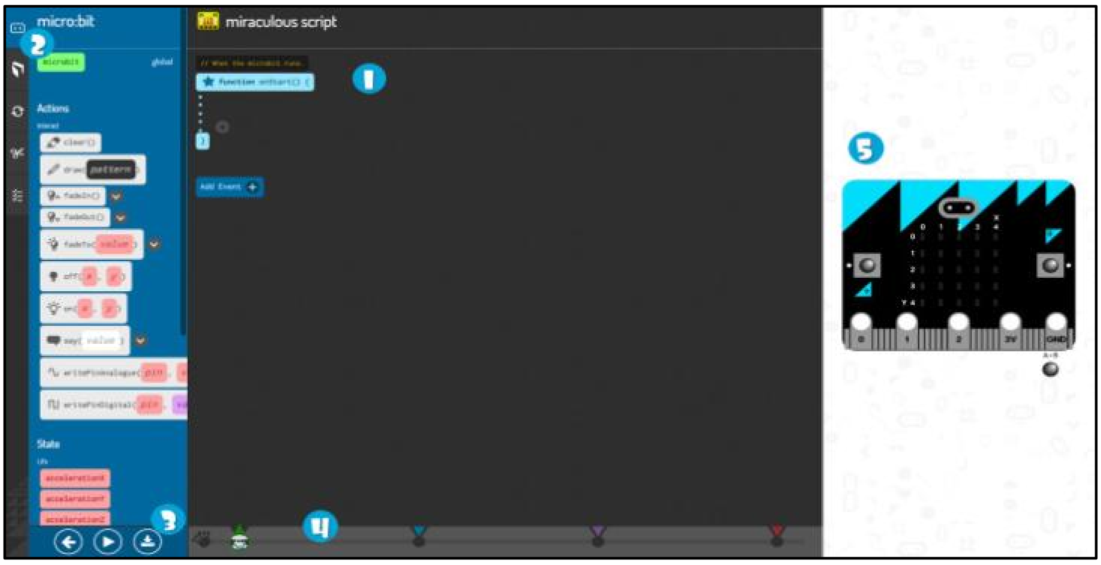
Accessing the editor

The CK JavaScript editor is best accessed using Google Chrome as your web browser.

1. Visit: microbit.co.uk/create-code

2. Select 'New Project' under the CK JavaScript editor

Tour of the Code Kingdoms JavaScript editor



What is it?	What is it for?
1. Main coding window	Writing code! This is where you can drag blocks of code or use a text input to write programs.
2. Menu	a. micro:bit – access code chunks to control the behaviour of the micro:bit b. Library – Maths, Random and Globals code libraries c. Language – access to loops and conditionals d. Snippets – copy and save snippets of code for later use e. Tutorial – guidance to complete the coding activity

3. Action buttons	Run and compile your code and return to My Scripts.
4. Slider	Transitioning between drag-and-drop code and text-based input to support learner progression. Also a useful tool for motivating learners and differentiating activities.
5. micro:bit simulator	Test out your code on the micro:bit simulator before compiling it to your device.

Tutorial System

The editor's in-built tutorial system guides you through the recipes in this booklet. The guidance for each recipe is found under the tutorial tab:

✓☰

The first three recipes have tutorials with close guidance, whereas the second three place greater emphasis upon discovering how to complete the recipe yourself.

Comments within code

Part of the tutorial system includes comments which aren't compiled as code but are designed to provide additional guidance. Comments are signalled by two forward slashes at the beginning of the line:

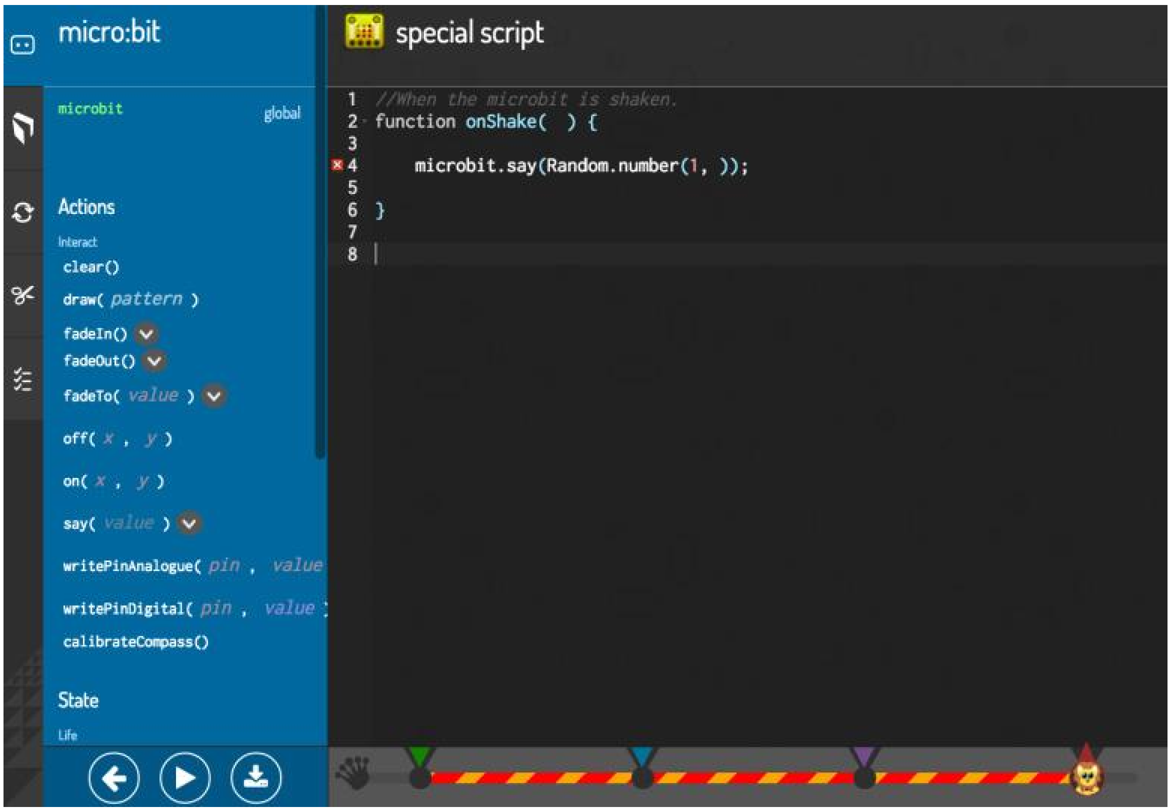
E.g. // this is a comment

You can add comments to your code by using the 'raw code' setting of the slider, which allows you to type the comment you wish to include.

Error Checking System

The error checking system helps to alert you to errors in your code and how to fix them, particularly when transitioning to using the text-based input.

You are alerted to incorrect code by the slider turning red and yellow, and a red cross next to the line of code that requires attention, as shown in the image below.



```
1  //When the microbit is shaken.
2  function onShake(  ) {
3
4      microbit.say(Random.number(1, ));
5
6  }
7
8
```

In this instance, there is a number missing from inside the brackets on line 4.

Learning outcomes

The learning outcomes are taken from the Computing at School Progression Pathways document. Each outcome is constructed using the colour of the progression level and the strand of learning it relates to.
E.g. the code OA2 = Orange / Algorithms / Statement 2.

	Recipe Name	Learning Outcomes
1	Roll the dice	• I can create a simple program in a visual language. (PP1) • I can execute, check and change a program. (PP2) • I understand that programs execute by following precise instructions. (PP3) • I use care and precision to avoid errors. (PA3) **Extension**: Display the number as it would be displayed on a die. • I can design a simple algorithm using selection (if) statements. (YA2) • I can use logical reasoning to predict outcomes. (YA3)
2	Mood swing	• I understand that algorithms are run on digital devices as programs. (YA1) • I can design a simple algorithm using loops and if statements. (YA2) • I can use arithmetic operators, if statements, and loops, within programs. (YP1) • I can detect and correct errors (debug) my programs. (YP3) **Extension**: While the micro:bit is sad, lower the brightness, and increase it if the micro:bit is happy. • I can use logical reasoning in conditions and am aware of inputs. (OA3) • I can create programs that implement a solution to achieve given goals. (OP1)

	Recipe Name	Learning Outcomes
3	Flashpoint	• I can design solutions that use repetition and if, then and else. (OA1) • I can use logical reasoning to predict outcomes. (YA3) • I can declare and assign variables. (OP2) • I can detect and correct errors (debug) my programs. (YP3) **Extension**: Implement a best of 3 scoring system for players • I can create programs that implement a solution to achieve given goals. (OP1) • I can use a (post-tested) while loop, and if, then (and else) statements. (OP3)
4	Worst. Snake. Ever.	• I can design solutions that use repetition and if, then and else. (OA1) • I can use logical reasoning in conditions and am aware of inputs. (OA3) • I can create programs that achieve given goals. (OP1) • I can declare and assign variables. (OP2) • I can use a (post-tested) while loop, and if, then (and else) statements. (OP3) **Extension**: Implement wrapping around the edges so the player goes to other side • I understand when to use if, and if, then and else statements. (BP1) • I can decompose a problem into parts and create sub-solutions for them. (BA2)

Recipe Name | **Learning Outcomes**

5 Quick-finger

- I can design solutions that use repetition and if, then and else. (OA1)
- I can use logical reasoning in conditions and am aware of inputs. (OA3)
- I can use variable and relational operators to govern termination of a loop. (BP2)
- I can use a (post-tested) while loop, and if, then (and else) statements. (OP3)

Extension: Implement up/down arrow into game which requires tilting

- I can decompose a problem into parts and create sub-solutions for them. (BA2)
- I recognise that different solutions exist for the same problem. (BA3)

6 Maze runner

- I understand that iteration is the repetition of a process such as a loop. (PuA1)
- I can represent a solution using a structured notation. (PuA3)
- I understand when to use if, and if, then and else statements. (BP1)
- I can use a variable and relational operators to govern termination of a loop.(BP2)
- I can use a (post-tested) while loop, and if, then (and else) statements. (OP3)

Extension: Implement tilting forward and back between additional maze levels

- I can identify patterns in situations and can use these to solve problems (PuA4)
- I can decompose a problem into parts and create sub-solutions for them. (BA2)

Code Kingdoms, Makewav.es & Open Badges

Using the Recipes created by Code Kingdoms for the BBC micro:bit editor, upload your completed program either as a screenshot or using the HEX file directly. You can then claim your Open Badge through the Makewav.es platform – all of the badges can be found here.

Recipe #1 – By the end of this activity, we will be able to shake a random number out of our micro:bit!

Create simple dice using random numbers.

Recipe #2 – By the end of this activity, we will have a micro:bit that is sad when it is tilted away, and happy when it is facing us!

Create a program that makes the micro:bit's mood swing!

Recipe #3 – By the end of this activity, we will have a micro:bit that will start a 2-player game when shaken.
The aim of the game is to be the first to press the button when all the lights turn on. Are your reflexes on point?

Recipe #4 – By the end of this activity, we will have a flashing dot (the snake) moving around the screen (when micro:bit is tilted) catching the static dots (the apples). Unlike the old mobile phone game, the snake doesn't grow as it feeds, making it the worst snake ever!

Recipe #5 – By the end of this activity, we will have a single player reaction game. The player has to quickly (and correctly) press button A or B based on the arrow that appears on screen; they receive a score after 10 rounds.

Recipe #6 – By the end of this activity, we will have a game that requires the player to tilt the micro:bit to guide their flashing dot through a maze drawn on the LED screen.

What is Makewav.es?

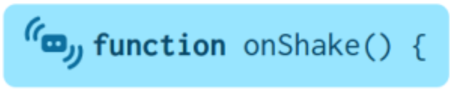
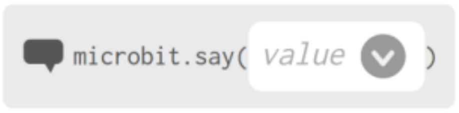
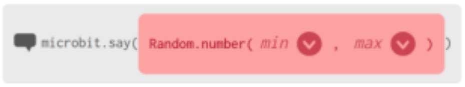
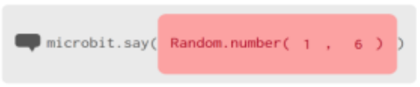
Makewaves is the safe social badging platform for schools that enables pupils to easily and safely earn and display Open Badges. Join the Makewaves badging community to enable your pupils to access a range of exciting badge challenges from a wide mix of partner organisations that range from STEM organisations such as the Royal Observatory Greenwich, health and wellbeing such as NHS England, and coding and computing, including Code Kingdoms.

Makewaves is free to join and includes free telephone training and support. www.makewav.es/computingcurriculum/join

To find out more about about Open Badges, go to openbadges.org.

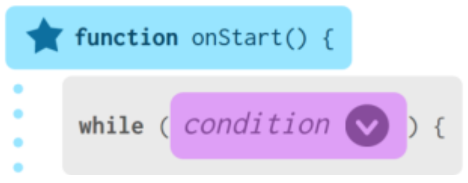
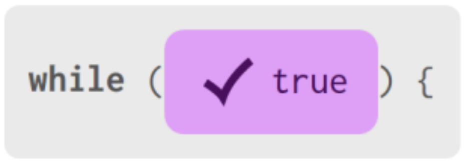
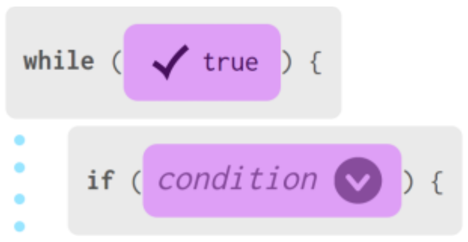
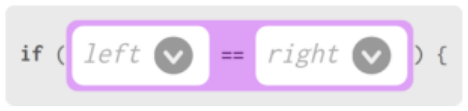
Recipe 1: Roll the Dice!

By the end of this activity, we will be able to shake a random number out of our micro:bit.

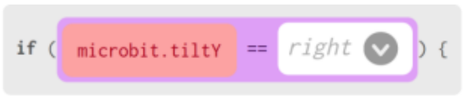
1. Click the `Add` button, and select `onShake` .	 function onShake() {
Why? *We need the micro:bit to tell us when it's been shaken*	
2. Navigate to the micro:bit tab in the left sidebar.	
Where? *The micro:bit tab is the first button at the top of the sidebar*	
3. Drag the `say` chunk from the menu so that it is within the `onShake` event.	 microbit.say(value)
Hint: *When dragging, wait for the black and orange placeholder to appear before dropping*	
4. Navigate to the Library tab in the left sidebar.	
Where? *The Library tab has the book icon in the sidebar*	
5. Choose the Random library.	
6. Drag `number(min, max)` from the menu so that it replaces the `value` in the `say` chunk of program.	 microbit.say(Random.number(min , max))
7. Click on the arrow next to `min` and enter 1. Repeat this step for `max` , entering the number 6.	 microbit.say(Random.number(1 , 6))

Recipe 2: Mood Swings

By the end of this activity, we will have a micro:bit that is sad when it is tilted away, and happy when it is facing us!

1. Navigate to the Language tab in the left sidebar.	
***Where?** The Language tab has the loop icon in the sidebar*	
2. Drag the `while` chunk from the sidebar until it is part of the `onStart` event.	 `function onStart() {` `while ( condition ) {`
3. Click on the `condition` menu inside the `while` chunk, and choose the `true` option at the top of the list.	 `while ( ✓ true ) {`
***Why?** A while-true loop is used when we want to repeat something forever. In our case, this means until the micro:bit is reset or runs out of power. Since the micro:bit will never stop smiling or frowning in this recipe, we use a while-true loop to make sure the program runs continuously by repeating forever.*	
4. Drag an `if` chunk from the sidebar until it is part of the while loop.	 `while ( ✓ true ) {` `if ( condition ) {`
***Hint:** When dragging, wait for the black and orange placeholder* ■ *to appear before dropping*	
5. Click on the `condition` menu inside the `if` chunk, and choose the `left == right` option.	 `if ( left == right ) {`
***Why?** We want to check whether the micro:bit's tilting sensor (accelerometer) is detecting that it is being held level*	
6. Navigate to the micro:bit tab in the left sidebar.	

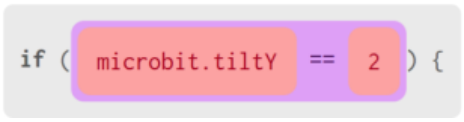
7. Scroll down until you find the **tiltY** chunk, and drag it into the **left** part of the if condition.



```
if ( microbit.tiltY == right ) {
```

Why? The micro:bit can check whether it is being tilted side to side (the x-direction) or forwards and backwards (the y-direction). In this recipe, we will be checking the y-direction, so we use the tiltY chunk instead of tiltX.

8. Click on the arrow in the **right** part of the if condition, choose a **number** value and enter 2.

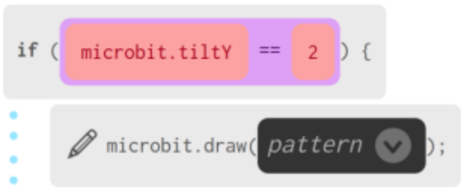


```
if ( microbit.tiltY == 2 ) {
```

Why? The micro:bit expresses how much it is tilted by a number. In the x-direction (side to side), the lefthand side of the screen is given the number 0 and the far righthand side is given the number 4. This means the middle column actually has the number 2. The same is true for the y-direction (top to bottom); the top ‘row’ of the screen is given the number 0, and the bottom ‘row’ the number 4, which means the middle ‘row’ has the number 2.

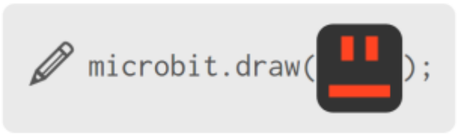
9. Navigate to the micro:bit tab in the left sidebar.

10. Drag the **draw** chunk from the sidebar until it is part of the **if** chunk we have just made.



```
if ( microbit.tiltY == 2 ) {
    microbit.draw( pattern );
```

11. Click on the **pattern** menu inside the **draw** chunk and choose the neutral face.



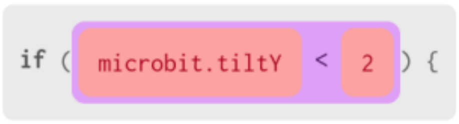
```
microbit.draw( );
```

12. Save the **if** chunk as a snippet by dragging the **if** chunk into the bottom half of the left sidebar.

13. Drag this saved snippet out of the sidebar into the while loop. Do this once more. You should now have three consecutive **if** chunks inside the while loop.

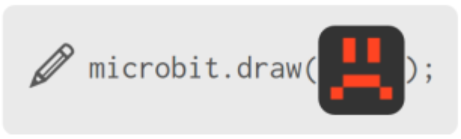
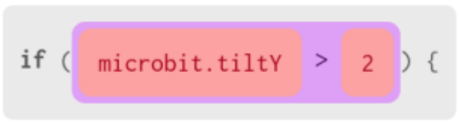
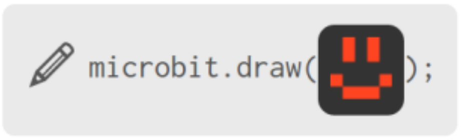
Why? We need three if chunks because we need to check whether the micro:bit is being tilted forward, back, or not at all

14. Click on the **condition** menu inside the second **if** chunk, and choose the **left < right** option.



```
if ( microbit.tiltY < 2 ) {
```

Why? This if chunk will deal with the case when the micro:bit is tilted away from the player

15. Click on the **pattern** menu inside this **draw** chunk and choose the sad face.	 `microbit.draw( );`
16. Click on the **condition** menu inside the third **if** chunk, and choose the left > right option.	 `if ( microbit.tiltY > 2 ) {`
Why? *This if chunk will deal with the case when the micro:bit is tilted towards the player*	
17. Click on the **pattern** menu inside this **draw** chunk and choose the happy face.	 `microbit.draw( );`

Recipe 3: Flashpoint

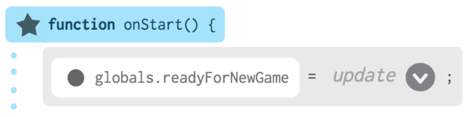
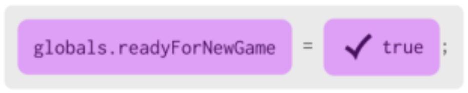
By the end of this activity, we will have a micro:bit that will start a 2-player game when shaken. The aim of the game is to be the first to press the button when all the lights turn on. Are your reflexes on point?

This may sound daunting at first, but programmers like to break down big, hard problems into smaller, more manageable problems. Once that's done, all we need to do is solve the smaller problems and put them all together.

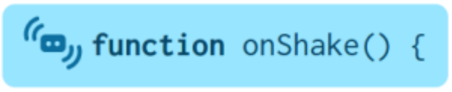
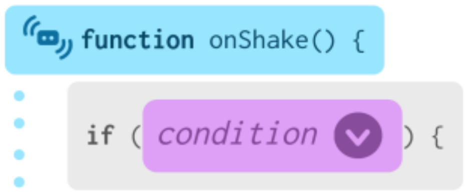
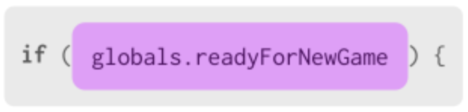
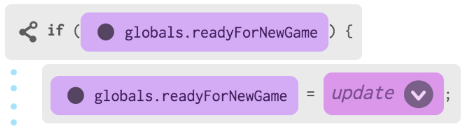
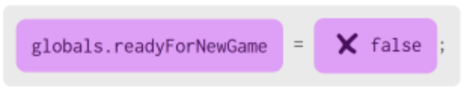
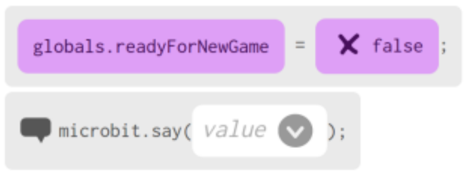
In this game, there are four parts we need to figure out:
1. what we need to keep track of throughout the game;
2. what to do when the micro:bit is shaken;
3. what to do when Player 1 presses their button; and
4. what to do when Player 2 presses their button.

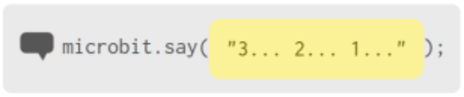
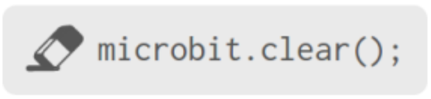
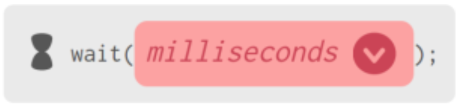
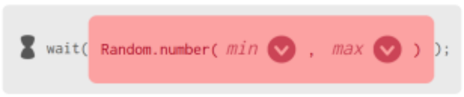
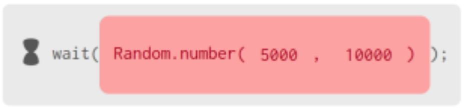
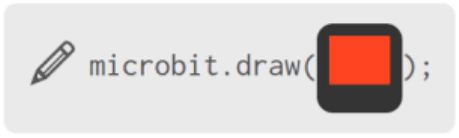
All of these together make up the whole game.

Let's start with setting up the game.

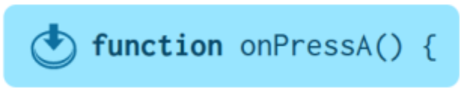
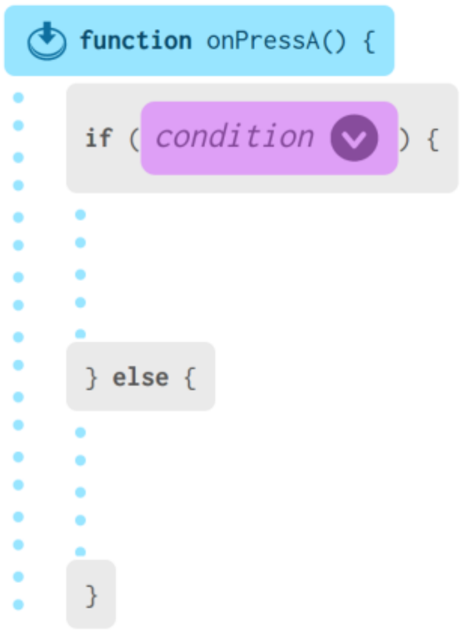
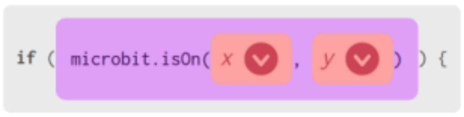
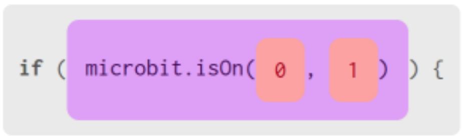
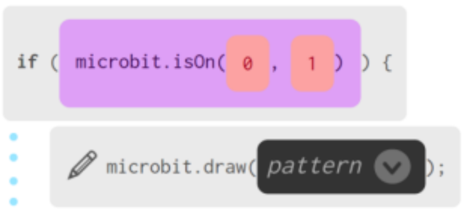
1. Navigate to the Library tab in the left sidebar and select the globals section.	
***Where?** The Library tab has the book icon in the sidebar*	
2. Click the icon, and give the variable the name `readyForNewGame`.	readyForNewGame
***Why?** We need to keep track of when a game is being played, otherwise the game would restart in the middle of a game if the micro:bit is shaken. Because this value can change at any time, we call it a variable, and since we want to be able to change it from anywhere in the program, we call it a <u>global</u> variable. Make sure your variables have descriptive names!*	
3. Drag the `readyForNewGame` chunk from the sidebar until it is part of the `onStart` event.	 `function onStart() {` `globals.readyForNewGame = update ;`
4. Click on the `update` menu next to the `readyForNewGame` chunk, and choose Boolean, and then the `true` option at the top of the list.	 `globals.readyForNewGame = ✓ true ;`
***Why?** When the program first starts, we are ready to start the game.This will change once the game is started.*	

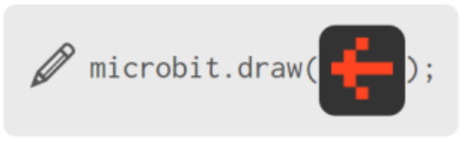
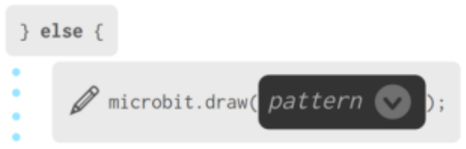
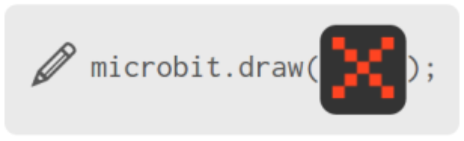
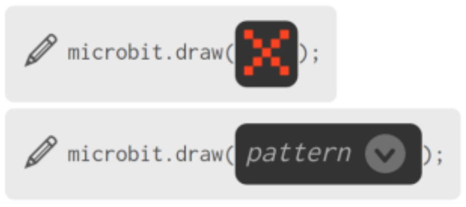
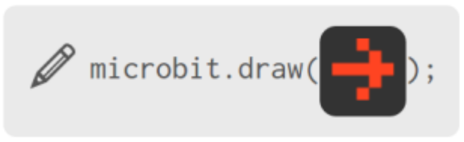
Now let's tell the micro:bit what to do if shaken.

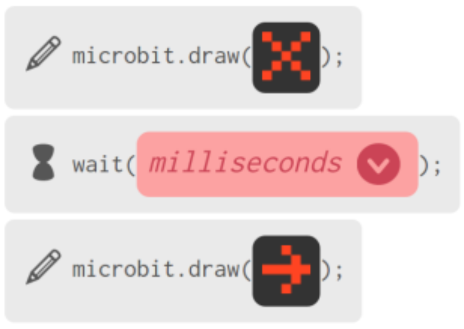
1. Click the **Add** button, and select **onShake** .	 `function onShake() {`
Why? *We need the micro:bit to tell us when it's been shaken.*	
2. Navigate to the Language tab in the sidebar.	
3. Drag an **if** chunk from the sidebar so it is within the onShake event.	 `function onShake() {` `if ( condition ) {`
4. Click on the **condition** menu inside the **if** chunk, and choose the **readyForNewGame** variable in the bottom section of the menu.	 `if ( globals.readyForNewGame ) {`
Why? *This will check whether the readyForNewGame variable is set to true before running the rest of our program.*	
5. Navigate to the Library tab in the sidebar.	
6. Choose the Globals library.	
7. Drag the **readyForNewGame** chunk from the sidebar until it is part of the **if** chunk.	 `if ( globals.readyForNewGame ) {` `globals.readyForNewGame = update ;`
8. Click on the **update** menu next to the **readyForNewGame** chunk, and choose the **false** option from the list.	 `globals.readyForNewGame = false ;`
Why? *Now that we're starting a game, we can set the value of readyForNewGame to be false so that no new games can start*	
9. Navigate to the micro:bit tab in the sidebar.	
10. Drag the **say** chunk from the menu into the **if** chunk.	 `globals.readyForNewGame = false ;` `microbit.say( value );`

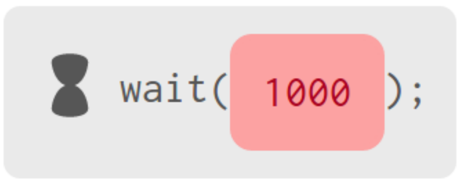
11. Change the `value` inside the `say` chunk to: "3… 2… 1…"	 `microbit.say( "3... 2... 1..." );`
12. Drag the `clear` chunk from the sidebar under `say` .	 `microbit.clear();`
13. Navigate to the Language tab in the sidebar.	
14. Drag the `wait` chunk from the sidebar under `clear` .	 `wait( milliseconds );`
15. Navigate to the Library tab in the left sidebar.	
16. Choose the Random library.	
17. Drag `number(min, max)` from the menu so that it replaces the `millisecond` in the `wait` chunk.	 `wait( Random.number( min , max ) );`
18. Click on the arrow next to `min` and enter 5000. Repeat this step for `max`, entering the number 10000.	 `wait( Random.number( 5000 , 10000 ) );`
Why? *We want the micro:bit to wait a different amount of time in each game (otherwise it would be boring!) between 5 and 10 seconds. Note that wait time is given in milliseconds, so half a second is 500, one second is 1000, and so on.*	
19. Navigate to the micro:bit tab in the sidebar.	
20. Drag the `draw` chunk from the sidebar to under the `wait` chunk, choosing all the lights to be turned on by clicking on the `pattern` .	 `microbit.draw( 🟥 );`

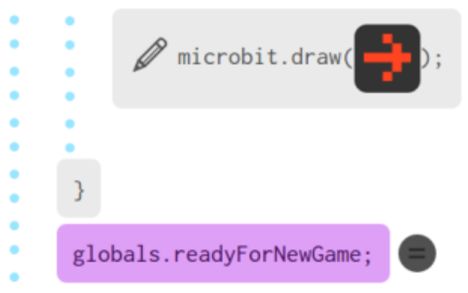
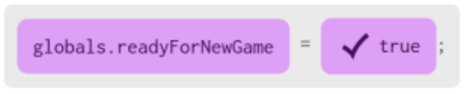
Now all that's left is to let the micro:bit know what to do when the buttons are pressed.

1. Click the **Add** button, and select **onPress** .	 `function onPressA() {`
2. Drag the **if** chunk and the **else** chunk from the Language tab into the **onPress** event.	 `function onPressA() {` `if ( condition ) {` `} else {` `}`
3. Navigate to the micro:bit tab in the sidebar.	
4. Drag **isOn(x, y)** from the lower part of the sidebar so that it replaces the **condition** in the **if** chunk.	 `if ( microbit.isOn( x , y ) ) {`
5. Click on the arrow next to **x** and enter the number 0. Repeat this step for **y** , entering the number 1.	 `if ( microbit.isOn( 0 , 1 ) ) {`
Why? If the LEDs are on, the player has won, otherwise they've pressed the button too early and lost! We don't need to look at all the LEDs, since the one at the (0,1) position will only be turned on in this program when all the LEDs are on.	
6. Navigate to the micro:bit tab in the sidebar.	
7. Drag the **draw** chunk from the sidebar into the top half of the **if** chunk.	 `if ( microbit.isOn( 0 , 1 ) ) {` `microbit.draw( pattern );`

8. Click on the `pattern` menu inside this `draw` chunk and choose the arrow pointing to the left.	 `microbit.draw( );`
9. Drag a `draw` chunk from the sidebar into the bottom half of the `if` chunk.	 `} else {` `microbit.draw( pattern );`
10. Click on the `pattern` menu inside this `draw` chunk and choose the cross pattern.	 `microbit.draw( );`
11. Drag another `draw` chunk from the sidebar into the bottom half of the `if` chunk.	 `microbit.draw( );` `microbit.draw( pattern );`
12. Click on the `pattern` menu inside this `draw` chunk and choose the arrow pointing to the right.	 `microbit.draw( );`

13. Drag the `draw` chunk from the Language tab in between the two `draw` chunks in the bottom half of the `if` chunk.	 `microbit.draw( );`
	`wait( milliseconds );`
	`microbit.draw( );`

14. Click on the arrow next to the `millisecond` in the `wait` chunk, and type in 1000.	 `wait( 1000 );`
Why? *Button A is on the left of the micro:bit so the arrow will point towards the player when they've won and to the other player (on the right) if they lose. We add a wait so that there is a second for both players to see who has won.*	
15. Navigate to the Library tab in the sidebar.	

16. Choose the Globals library.	
17. Drag the readyForNewGame chunk from the sidebar until it is part of the onPress event, after the if chunk.	 microbit.draw( ); } globals.readyForNewGame;
18. Click on the update menu next to the readyForNewGame chunk and choose the true option at the top of the list.	 globals.readyForNewGame = ✓ true ;

Why? Now that we've finished a game, we can set the value of readyForNewGame to be true so new games can start.

For the B button, Click the Add button, select onPress and repeat the steps above, just reversing the direction of the arrows in Step 8 and Step 12.

Recipe 4: Worst. Snake. Ever

By the end of this activity, we will have a flashing dot (the snake) moving around the screen (when micro:bit is tilted) catching the static dots (the apples). Unlike the old mobile phone game, the snake doesn't grow as it feeds, making it the worst snake ever.

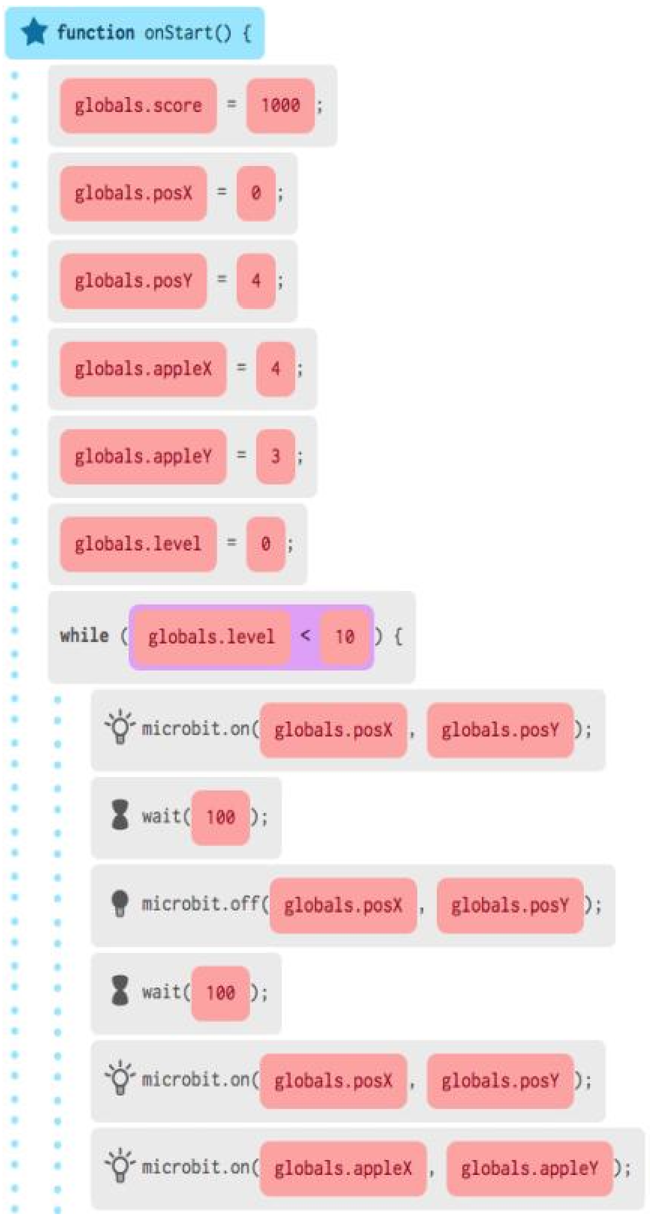
Section 1

1. We need a few variables to store certain values within the game. We need to give them initial values here. Score starts from a high number and decreases the longer the player plays (which means quicker times get higher scores); posX and posY keep track of the player's dot; appleX and appleY keep track of the apple; and level is the current level.

2. We don't want the game to run continuously, so we put a condition inside the while loop so that the loop only runs when the level variable is less than 10.

3. We need a way to differentiate between the player's dot and the apple. One way to do this is to flash the player's dot by turning it off and on again (with a small wait so that the player notices).

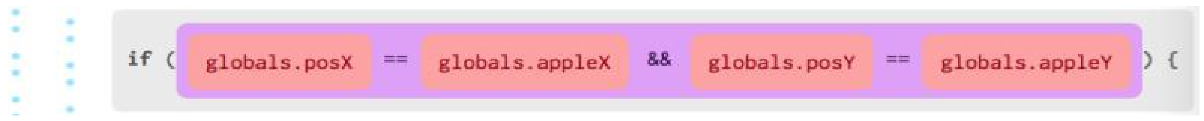
4. Refresh the screen to make sure that the LEDs for the player's dot and apple are on.



```
function onStart() {
    globals.score = 1000;
    globals.posX = 0;
    globals.posY = 4;
    globals.appleX = 4;
    globals.appleY = 3;
    globals.level = 0;
    while ( globals.level < 10 ) {
        microbit.on( globals.posX , globals.posY );
        wait( 100 );
        microbit.off( globals.posX , globals.posY );
        wait( 100 );
        microbit.on( globals.posX , globals.posY );
        microbit.on( globals.appleX , globals.appleY );
```

Section 2

1. If the player's dot has the same x- and y-coordinate, it means they're on top of each other, so the player has successfully collected the apple. Reward!

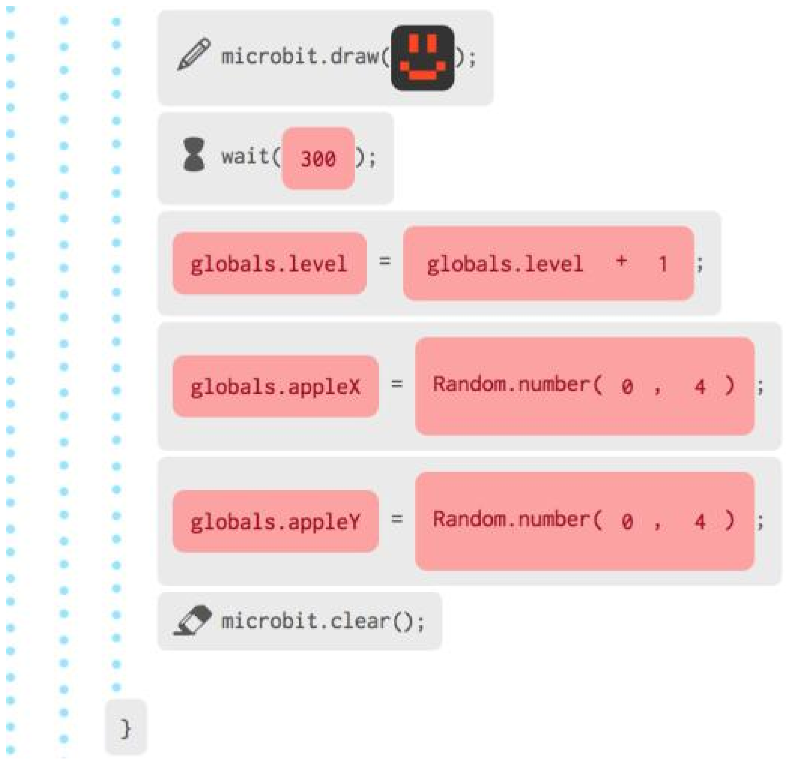


```
if ( globals.posX == globals.appleX && globals.posY == globals.appleY ) {
```

2. Update the 'levels' variable because the player has found the apple.

3. Pick a new, random position for the apple, ready for the next level.

4. Get rid of the old dots and reward smiley face.

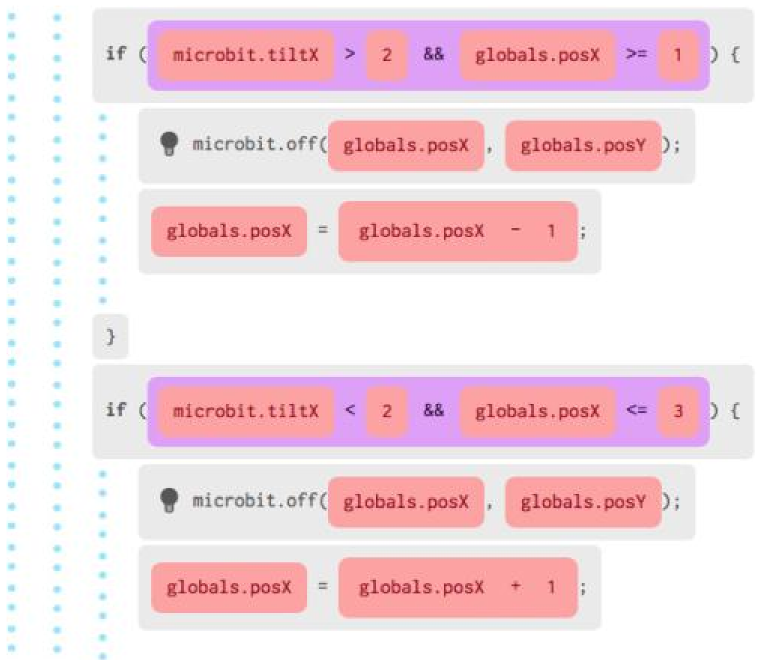


```
    microbit.draw(  );
    wait( 300 );
    globals.level = globals.level + 1 ;
    globals.appleX = Random.number( 0 , 4 ) ;
    globals.appleY = Random.number( 0 , 4 ) ;
    microbit.clear();
}
```

Section 3

1. If the micro:bit is being tilted to the left, turn off the old player dot, and update the posX variable to reflect the fact that it needs to move left.

2. If the micro:bit is being tilted to the right, turn off the old player dot, and update the posX variable to reflect the fact that it needs to move right.



```
if ( microbit.tiltX > 2 && globals.posX >= 1 ) {
    microbit.off( globals.posX , globals.posY );
    globals.posX = globals.posX - 1 ;
}
if ( microbit.tiltX < 2 && globals.posX <= 3 ) {
    microbit.off( globals.posX , globals.posY );
    globals.posX = globals.posX + 1 ;
```

Hint: Looking at these if chunks, you may see a pattern. This is where using the Snippets tab might be useful to allow you to easily reuse chunks of code you've written.

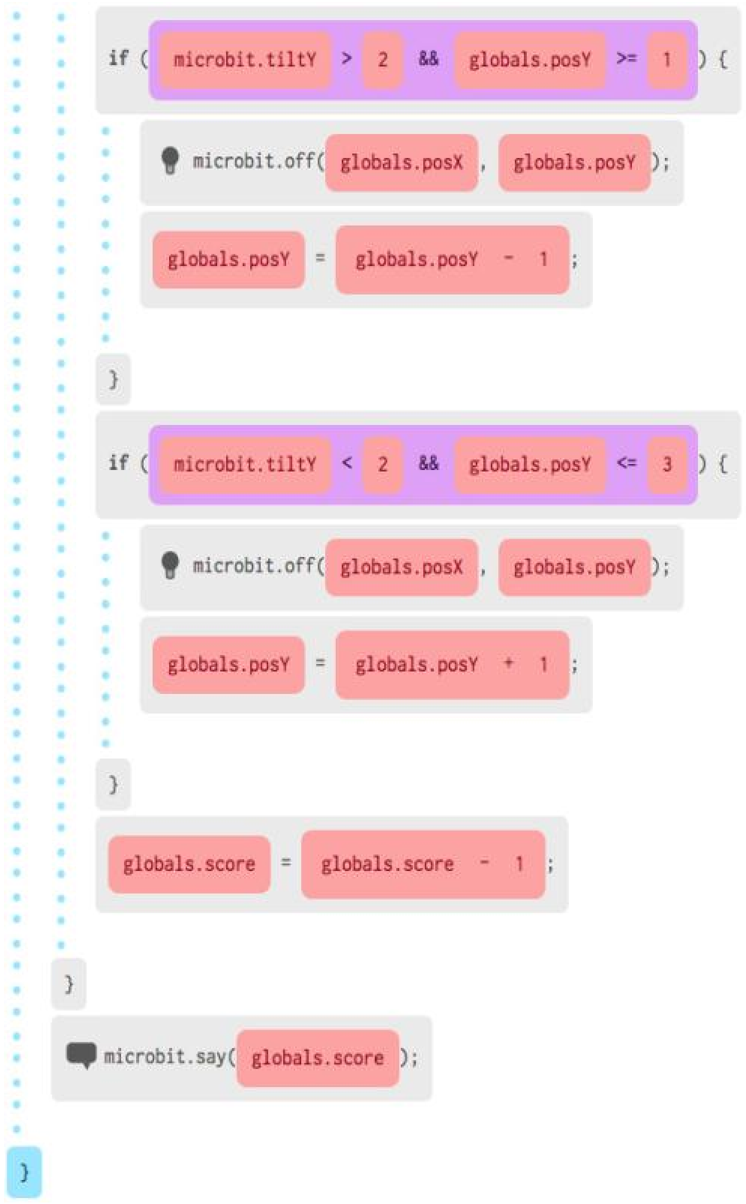
Section 4

1. If the micro:bit is being tilted down, turn off the old player dot, and update the posY variable to reflect the fact that it needs to move down.

2. If the micro:bit is being tilted up, turn off the old player dot, and update the posY variable to reflect the fact that it needs to move up.

3. Decrease the score by one every time the loop is run.

4. This is outside the while loop, which means the player must have picked up ten apples. So tell the player their score.



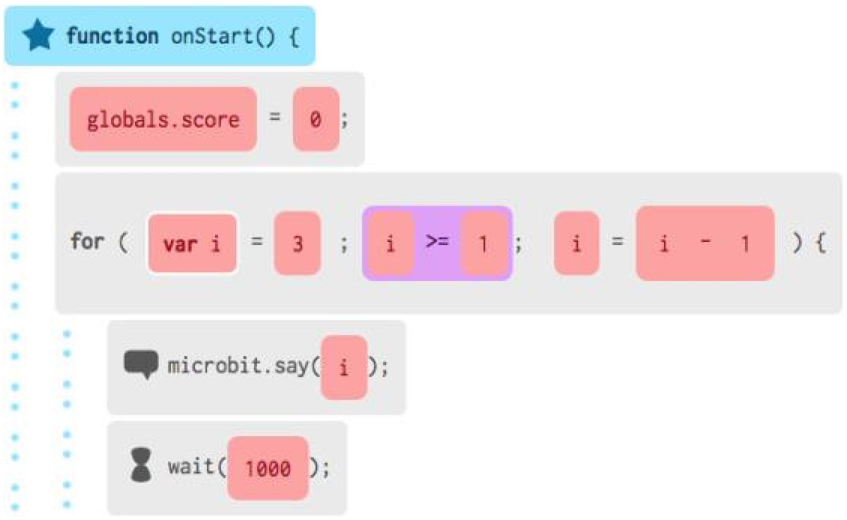
```
        if ( microbit.tiltY > 2 && globals.posY >= 1 ) {
            microbit.off( globals.posX , globals.posY );
            globals.posY = globals.posY - 1 ;
        }
        if ( microbit.tiltY < 2 && globals.posY <= 3 ) {
            microbit.off( globals.posX , globals.posY );
            globals.posY = globals.posY + 1 ;
        }
        globals.score = globals.score - 1 ;
    }
    microbit.say( globals.score );
}
```

Recipe 5: Quickfinger

By the end of this activity, we will have a single player reaction game. The player has to quickly (and correctly) press button A or B based on the arrow that appears on screen; they receive a score after 10 rounds.

Section 1

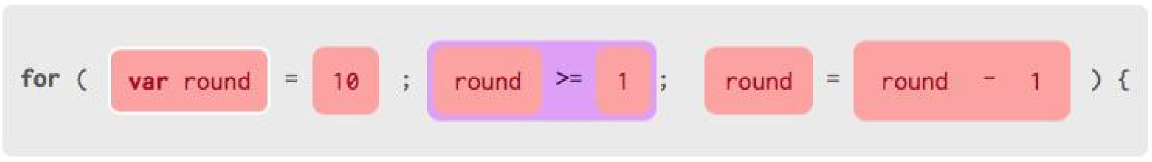
1. Create a variable to keep track of the score.
2. We want to repeat something a specific amount of times, so we use a 'for' loop with our loop counter 'i' going from 3 down to 1.
3. Display the loop counter variable with a 1 sec pause in between, so it becomes a countdown.



```
function onStart() {
    globals.score = 0 ;
    for ( var i = 3 ; i >= 1 ; i = i - 1 ) {
        microbit.say( i );
        wait( 1000 );
```

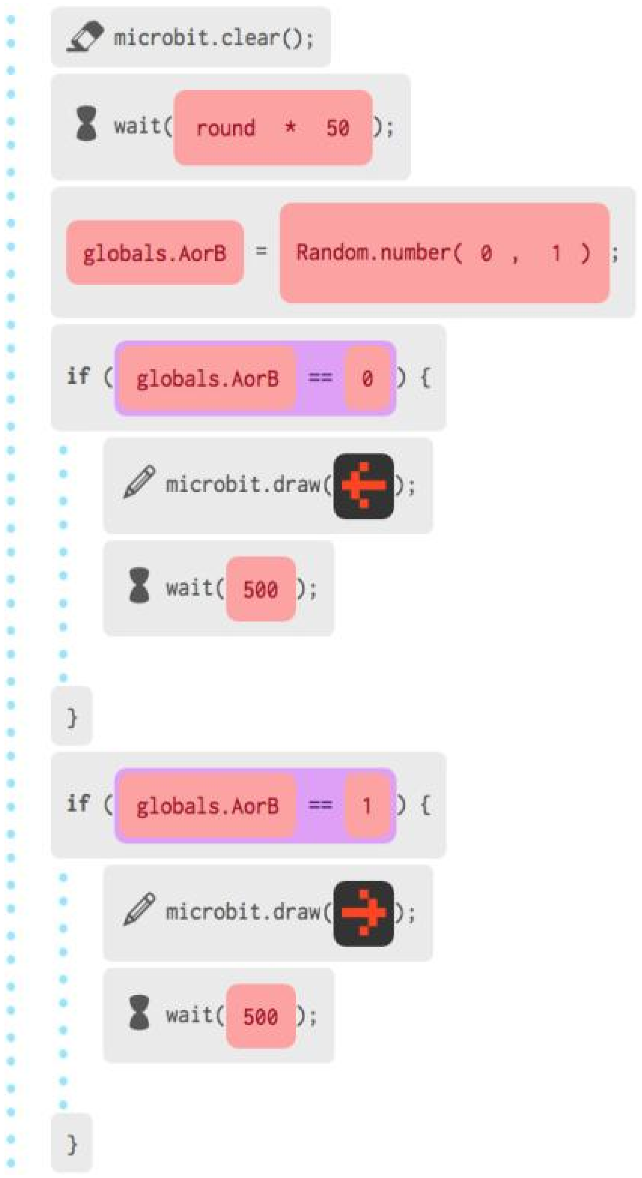
Section 2

1. The game lasts 10 rounds, so we use another ‘for’ loop using a loop counter called ‘round’.



```
for ( var round = 10 ; round >= 1 ; round = round - 1 ) {
```

2. Clear the screen.
3. Wait a little while – note the waiting time gets smaller as the rounds go by (as the loop counter ‘round’ will start from 10 and go down to 1).
4. Choose which direction to point randomly.
5. If it’s decided that it is to be 0, we say that’s left, and show the correct arrow accordingly.
6. If it’s decided that it is to be 1, we say that’s right, and show the correct arrow accordingly.



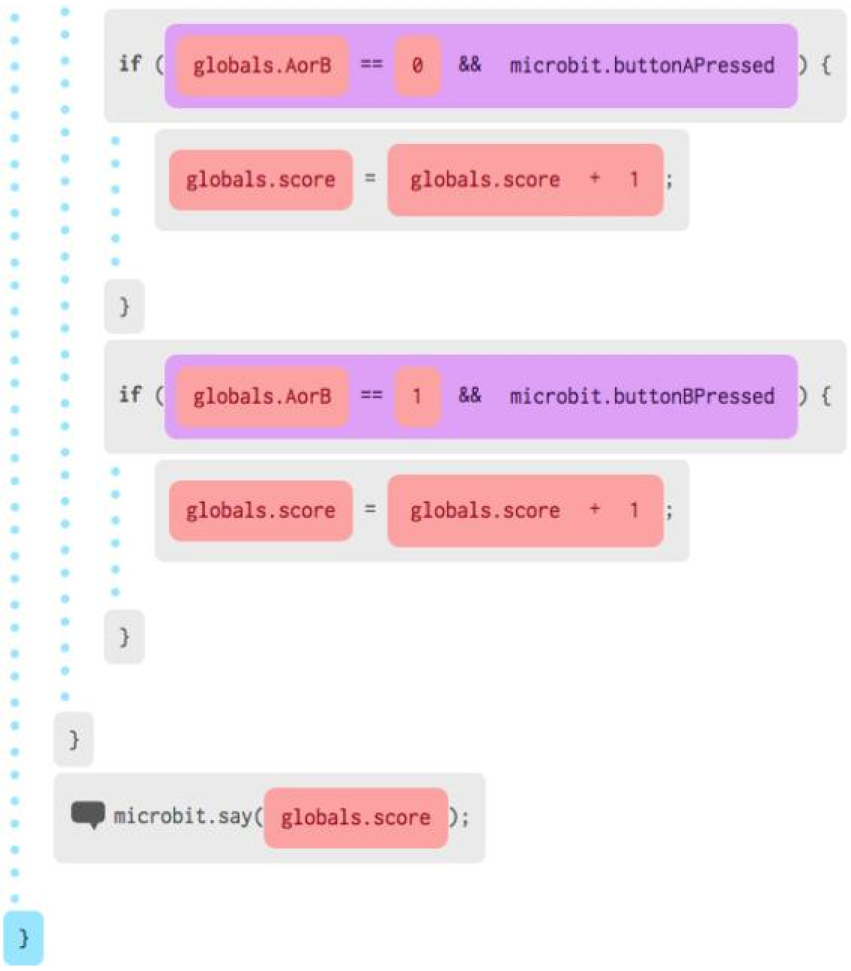
```
microbit.clear();
wait( round * 50 );
globals.AorB = Random.number( 0 , 1 ) ;
if ( globals.AorB == 0 ) {
    microbit.draw(←);
    wait( 500 );
}
if ( globals.AorB == 1 ) {
    microbit.draw(→);
    wait( 500 );
}
```

Section 3

1. If we displayed left, and the player pressed button A, give them a point by increasing their score by 1.

2. If we displayed right, and the player pressed button B, give them a point by increasing their score by 1.

3. We are outside of the ‘for’ loop, so we must have finished all ten rounds. Display the player’s score.



```
        if ( globals.AorB == 0 && microbit.buttonAPressed ) {
            globals.score = globals.score + 1 ;
        }
        if ( globals.AorB == 1 && microbit.buttonBPressed ) {
            globals.score = globals.score + 1 ;
        }
    }
    microbit.say( globals.score );
}
```

Recipe 6: Maze Runner

By the end of this activity, you will have a game that requires the player to tilt the micro:bit to guide their flashing dot through a maze drawn on the LED screen.

Section 1

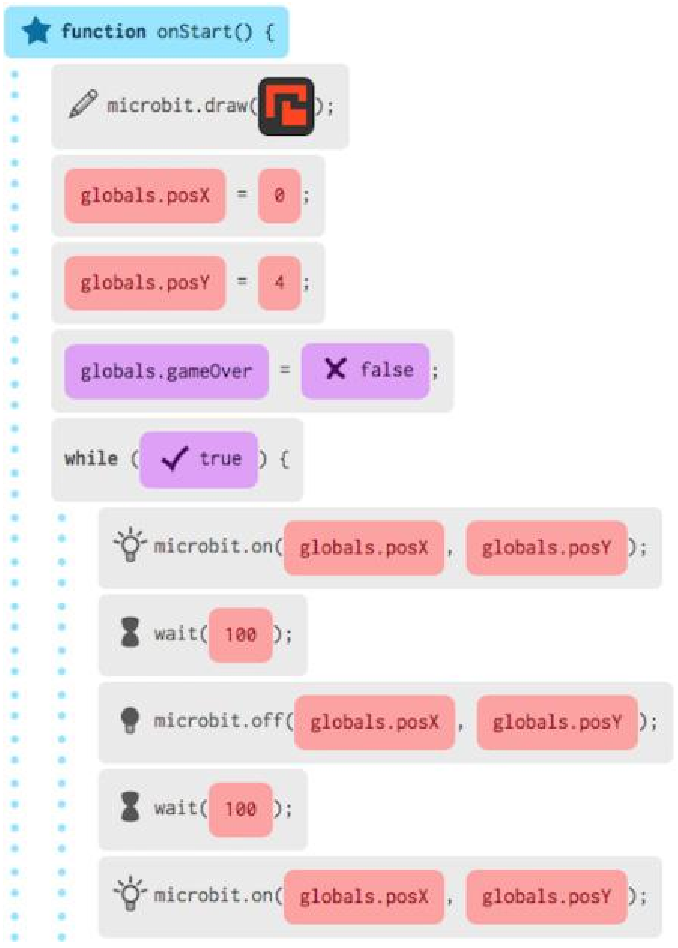
1. Draw the maze you want players to solve – be sure there is a place to start and a place to finish.

2. Store the player's current position in two variables, called posX and posY.

3. Set them to be wherever the player's dot should start.

4. The game will only end when the puzzle is solved, so use a while-true loop to keep the game running forever.

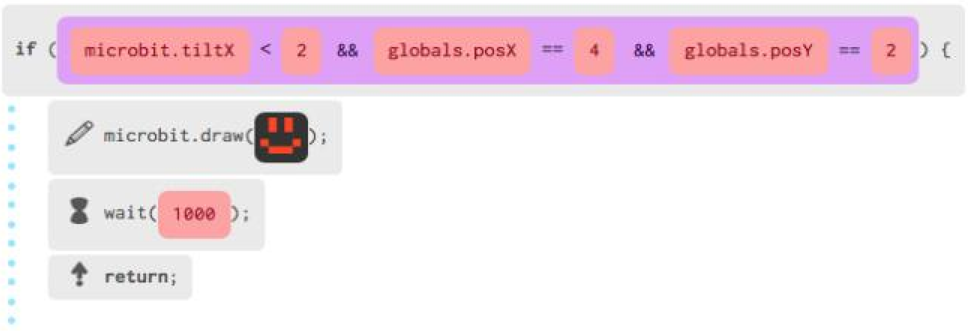
5. We need a way to differentiate between the maze and the player's dot (given by the posX and posY variables). One way to do this is to have the player's dot flashing, by turning it off briefly and then on.



```
function onStart() {
    microbit.draw( );
    globals.posX = 0 ;
    globals.posY = 4 ;
    globals.gameOver = false ;
    while ( true ) {
        microbit.on( globals.posX , globals.posY );
        wait( 100 );
        microbit.off( globals.posX , globals.posY );
        wait( 100 );
        microbit.on( globals.posX , globals.posY );
```

Section 2

1. If the micro:bit is being tilted in the right direction, and the player's dot is at the end of the maze, then the maze puzzle has been successfully completed. Draw a happy face and then end the program!



```
if ( microbit.tiltX < 2 && globals.posX == 4 && globals.posY == 2 ) {
    microbit.draw( );
    wait( 1000 );
    return;
```

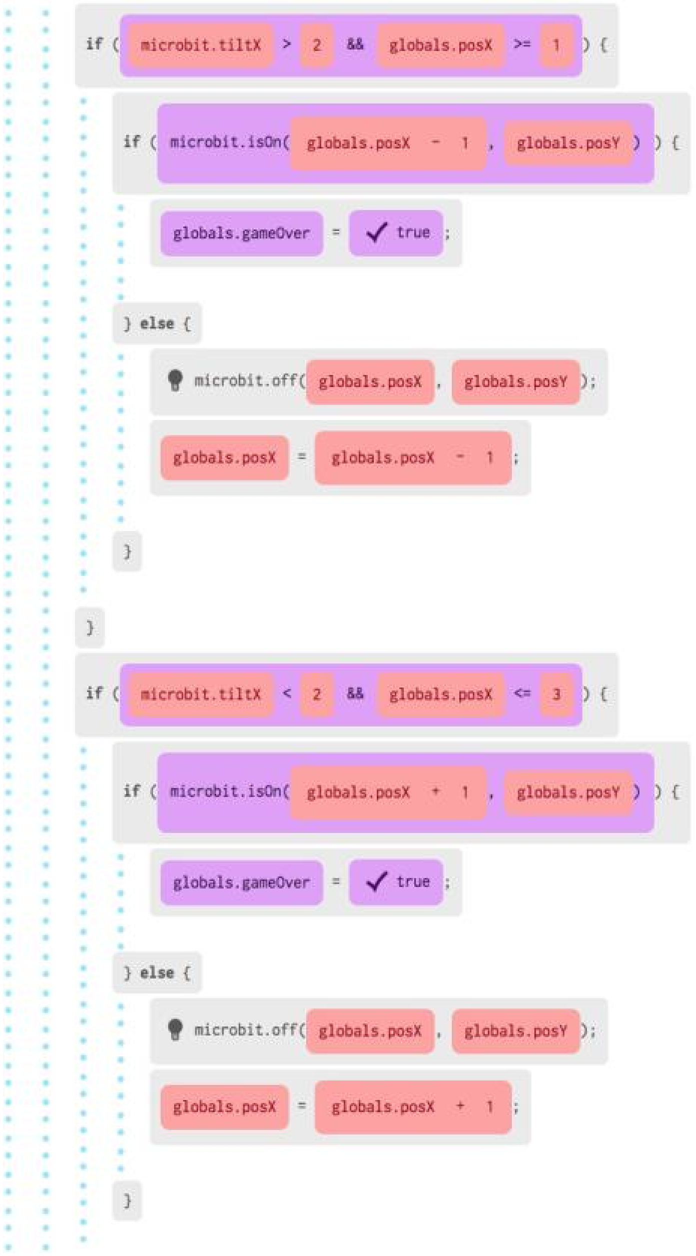
Section 3

1. If the micro:bit is being tilted to the left (and it's not on the edge), check that the LED at the point the player dot wants to move to is not on.

2. If it is, the player has hit a wall and the game is over (so turn the gameOver variable to true). Otherwise, update the player's dot position (given by posX and posY) to be one to the left of where it was. Turn the LED off at the player's old dot position.

3. If the micro:bit is being tilted to the right (and it's not on the edge), check that the LED at the point the player dot wants to move to is not on.

4. If it is, the player has hit a wall and the game is over (so turn the gameOver variable to true). Otherwise, update the player's dot position (given by posX and posY) to be one to the right of where it was. Turn the LED off at the player's old dot position.



```
if ( microbit.tiltX > 2 && globals.posX >= 1 ) {
    if ( microbit.isOn( globals.posX - 1 , globals.posY ) ) {
        globals.gameOver = true;
    } else {
        microbit.off( globals.posX , globals.posY );
        globals.posX = globals.posX - 1;
    }
}
if ( microbit.tiltX < 2 && globals.posX <= 3 ) {
    if ( microbit.isOn( globals.posX + 1 , globals.posY ) ) {
        globals.gameOver = true;
    } else {
        microbit.off( globals.posX , globals.posY );
        globals.posX = globals.posX + 1;
    }
```

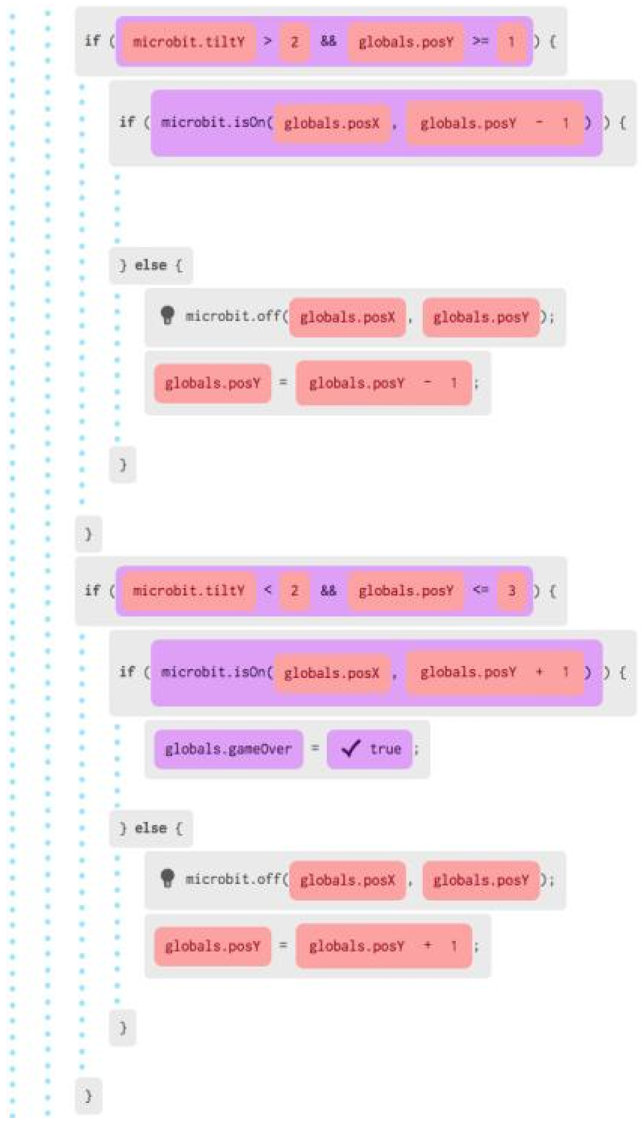
Section 4

1. If the micro:bit is being tilted up (and it’s not at the top), check that the LED at the point the player dot wants to move to is not on.

2. If it is, the player has hit a wall and the game is over (so turn the gameOver variable to true). Otherwise update the player’s dot position (given by posX and posY) to be one above where it was. Turn the LED off at the player’s old dot position.

3. If the micro:bit is being tilted down (and it’s not at the bottom), check that the LED at the point the player dot wants to move to is not on.

4. If it is, the player has hit a wall and the game is over (so turn the gameOver variable to true). Otherwise, update the player’s dot position (given by posX and posY) to be one below where it was. Turn the LED off at the player’s old dot position.

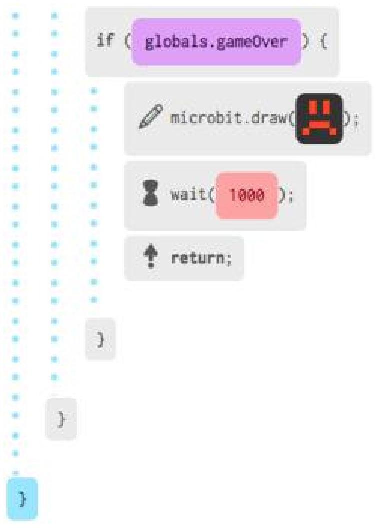


```
if ( microbit.tiltY > 2 && globals.posY >= 1 ) {
    if ( microbit.isOn( globals.posX , globals.posY - 1 ) ) {

    } else {
        microbit.off( globals.posX , globals.posY );
        globals.posY = globals.posY - 1 ;
    }
}
if ( microbit.tiltY < 2 && globals.posY <= 3 ) {
    if ( microbit.isOn( globals.posX , globals.posY + 1 ) ) {
        globals.gameOver = true ;
    } else {
        microbit.off( globals.posX , globals.posY );
        globals.posY = globals.posY + 1 ;
    }
}
```

Section 5

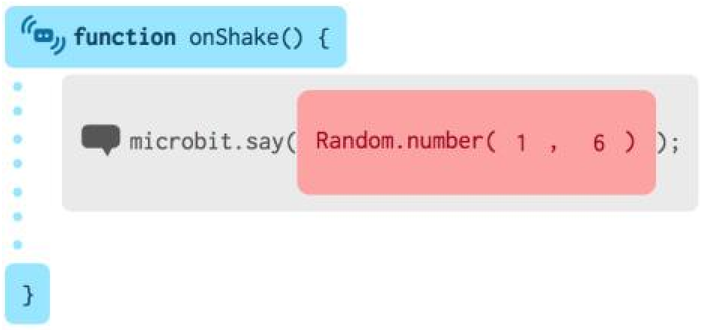
1. If, after all these checks about the tilting, the gameOver variable was set to true (because the player hit a wall), draw a sad face to let the player know and exit the program.



```
if ( globals.gameOver ) {
    microbit.draw( );
    wait( 1000 );
    return;
}
}
}
```

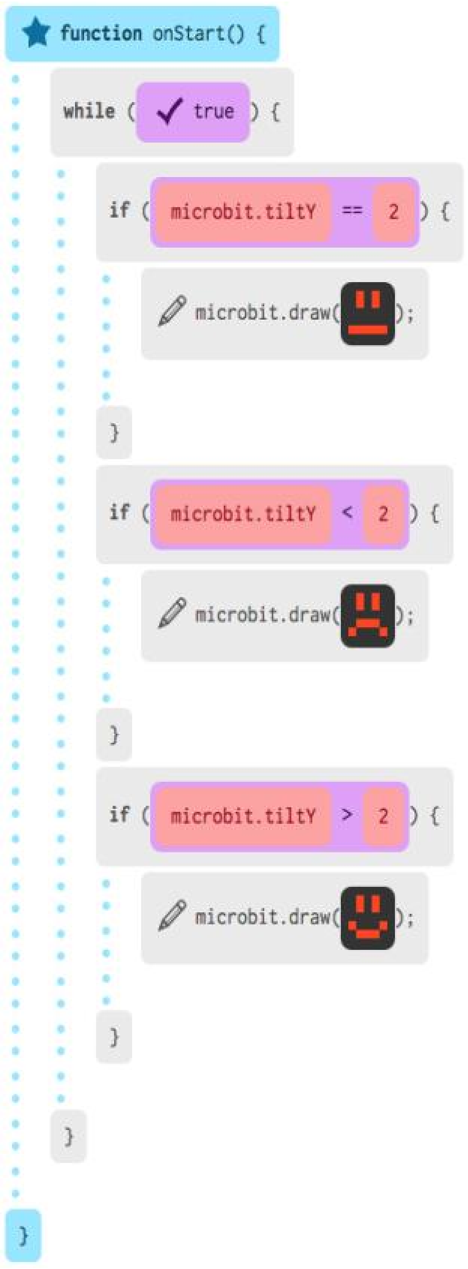
Challenges & Solutions

Recipe 1: Roll the Dice



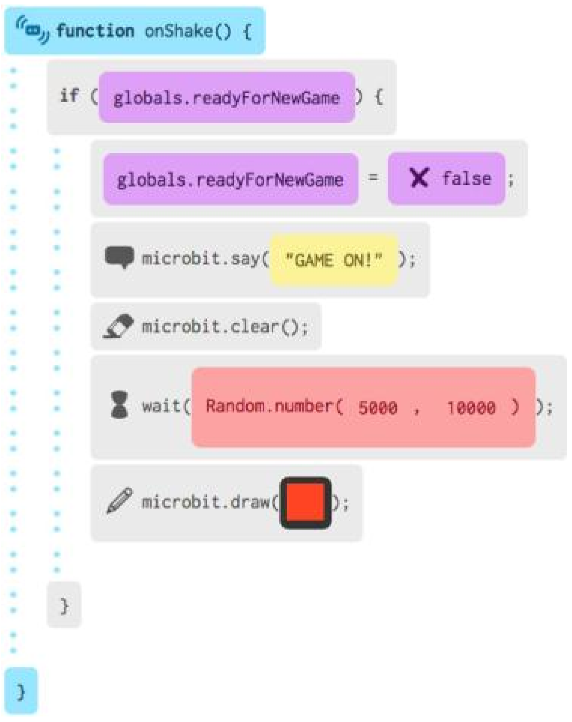
```
function onShake() {
    microbit.say( Random.number( 1 , 6 ) );
}
```

Recipe 2: Mood Swing

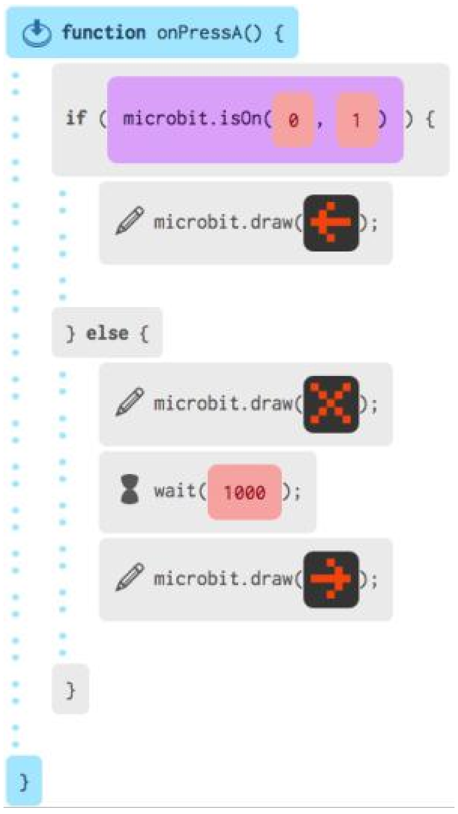


```
function onStart() {
    while ( true ) {
        if ( microbit.tiltY == 2 ) {
            microbit.draw( );
        }
        if ( microbit.tiltY < 2 ) {
            microbit.draw( );
        }
        if ( microbit.tiltY > 2 ) {
            microbit.draw( );
        }
    }
}
```

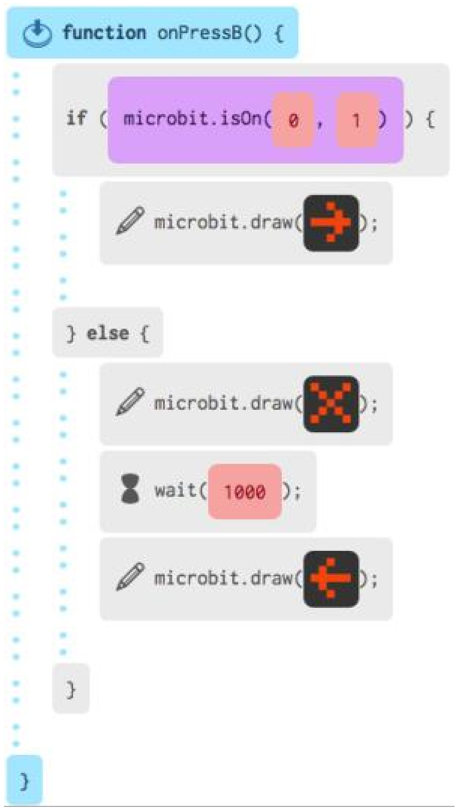
Recipe 3: Flashpoint



```
function onShake() {
    if ( globals.readyForNewGame ) {
        globals.readyForNewGame = ✕ false ;
        microbit.say( "GAME ON!" );
        microbit.clear();
        wait( Random.number( 5000 , 10000 ) );
        microbit.draw(🟥);
    }
}
```



```
function onPressA() {
    if ( microbit.isOn( 0 , 1 ) ) {
        microbit.draw(⬅);
    } else {
        microbit.draw(✖);
        wait( 1000 );
        microbit.draw(➡);
    }
}
```



```
function onPressB() {
    if ( microbit.isOn( 0 , 1 ) ) {
        microbit.draw(➡);
    } else {
        microbit.draw(✖);
        wait( 1000 );
        microbit.draw(⬅);
    }
}
```