

# Introduction to Computer Science with MakeCode for Minecraft

## Lesson 2: Events

In this lesson, we will learn about events and event handlers, which are important concepts in computer science and can be found in all programming languages. We will start with a fun unplugged activity that demonstrates cause and effect, and how events trigger actions in the real world. Next, we'll get hands-on with MakeCode in Minecraft, and finally, we'll challenge you to create your own MakeCode projects that use events to activate different parts of your program.

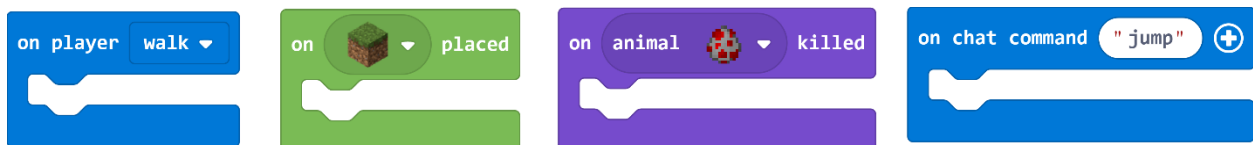
### Events

An "event" in computer science is an action or occurrence that is detected by a computer. For example, when someone clicks the button on their mouse, it generates a "mouse click event" for the computer. In real life, there are also events that may be associated with a following action, like Cause-and-Effect. Here are some examples:

Event		Action
It starts raining	➔	Open umbrella
The bell rings	➔	Students go to class
The Power button is pressed	➔	Computer turns on
Mouse button is clicked	➔	Open application

Can you think of some other Events and possible subsequent actions?

In programming, an event handler is a part of your program that runs when specific events happen (it "handles" the event). In MakeCode, these event handler blocks look like a square with a gap in the middle, and usually start with the word "on":



### Unplugged Activity: Events and Handlers

Teacher Note: An unplugged activity is an activity that takes place away from the computer, i.e., "unplugged" from a device. We usually try to introduce new concepts in a fun way that

gets kids up and moving, often reacting and interacting with other students face-to-face while playing a game or completing a challenge. Unplugged activities allow kids to practice concepts away from the computer so that when they move to activities on the computer, they have already walked through and thought about the concept on their own.

**Objective:**

To reinforce the event-driven programming model by acting out events and the resulting actions encapsulated by an event handler.

**Overview:**

Identify one-third of the class to be the "Events", and the rest of the class will be the "Event Handlers"

For the "Event" students, have them come up with an event to model. Students should invent their own events, but some examples could be:

- Door opens
- Lights turn off
- Clap hands twice
- Both hands raised

Once they've decided on their event, students should write down their event twice, on 2 different index cards. These cards should be shuffled and passed out to the "Event Handler" students.

Once the "Event Handler" students have received their assigned events, they should come up with some sort of action to take based on this event. Students should invent their own actions, but some examples could be:

- Door opens → Walk outside
- Lights turn off → Go to sleep
- Clap hands twice → Stomp feet 3 times
- Both hands raised → Say "Touchdown!"

Line up the "Event" students at the front of the classroom, and have each one perform their event. When the event is performed, the associated "Event Handler" students for that given event should also perform the subsequent action.

Once all the event and event handlers have been called, you can randomly call on different "Event" students to perform, and trigger different event handlers – you can speed this up and see if the students react quickly to their assigned event.

Materials:

- Index cards & pens/pencils

Rules:

- Unless instructed otherwise, students do not speak or make noise during this activity unless it is part of their event or action.
- Students should be aware of the activities of the other people in the classroom, but cannot tell other students what to do.

Reflections:

Have a discussion about how that felt/worked:

- Were there any programming errors or bugs in the system? Did a student miss handling an event?
- What was it like to keep track of the different events going on?
- Sometimes there was more than 1 event handler for a given event... how does that work?
- Could there be 1 event handler for multiple events? (yes)
- Could an event handler also trigger an event? (yes) If so, how would that work? (Lights turn off → Go to sleep → Teacher says "Wake up!")

Tips:

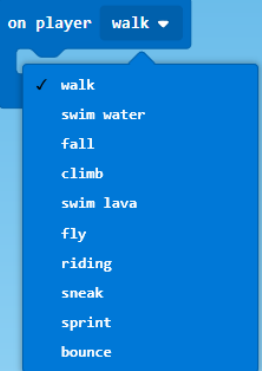
- SAFETY FIRST! Students, especially younger ones, can get quite silly with this and while it is meant to be fun and even funny, safety first!

Notes:

Computer programming connection: Computer programs are a set of instructions telling the computer how to process input and deliver output. An important part of programming is telling the computer WHEN to perform a certain task. Events are a way to trigger certain instructions.

## Activity: Yellow Brick Road

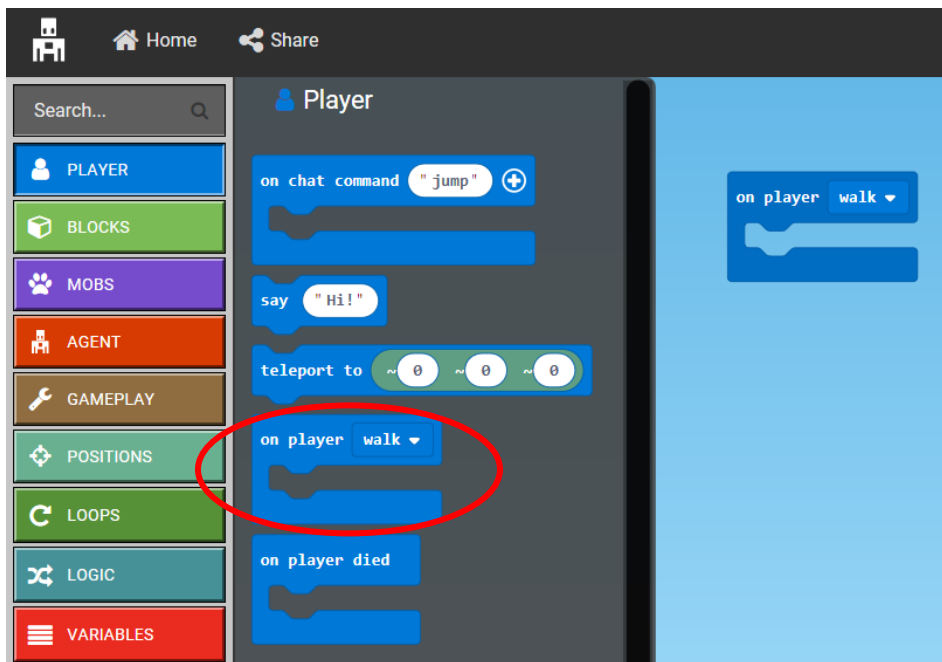
Teacher Note: A “birdhouse activity” is named after the birdhouses many of us made as our first project in wood shop class. Everybody in class follows the same instructions to make the same thing. Once everybody comes out with a birdhouse that looks halfway decent, you know they have all had at least an introduction to the concepts through an unplugged activity, and practiced new skills by making the birdhouse. We like to move from unplugged activities to birdhouses in preparation for more creative, open-ended projects once we know they have demonstrated the skills at least once. Just don’t stop after the birdhouse! Although it is easiest to assess skills with a birdhouse, be sure to give kids opportunities to apply those skills to meet more open-ended challenges.

	<p>The “On player walk” block is an event handler that looks for a specific action by the player. Its pull-down menu lists all sorts of actions a player might perform at any given time, such as walking, jumping, attempting to swim in lava, and more.</p>
--	---

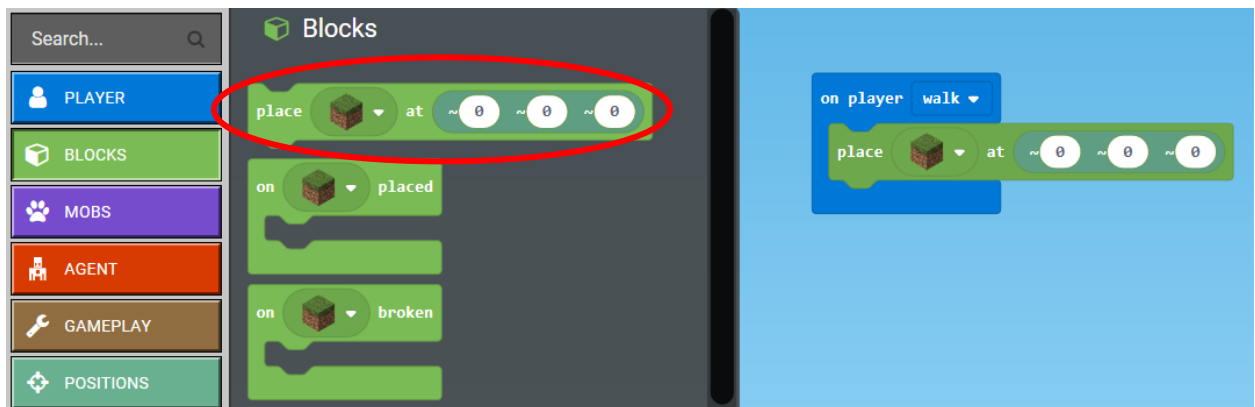
You can configure this event handler to cause something to happen when a player is walking. For example, you can leave flowers everywhere you walk!

### Steps:

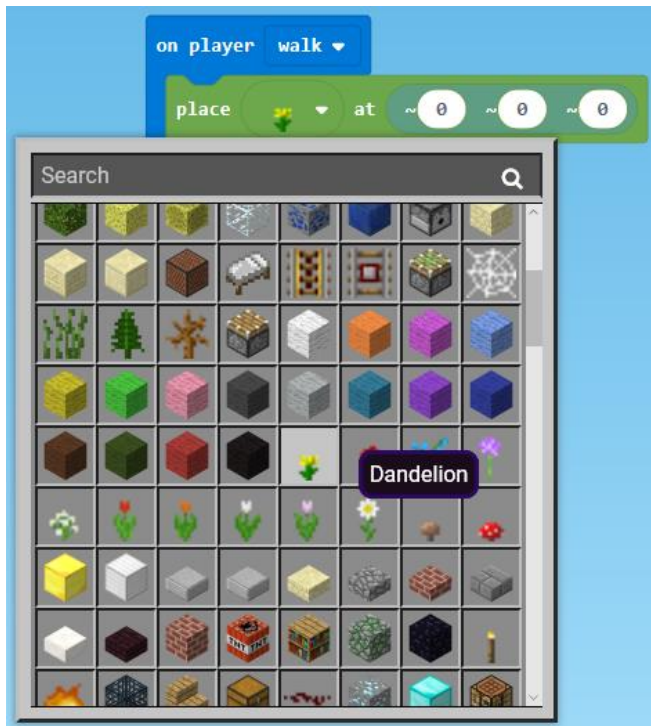
1. From the [Player](#) Toolbox drawer, drag the [On player walk](#) block into the coding Workspace.



2. From the **Blocks** Toolbox drawer, drag the **Place at** block under the **On player walk** block until you hear and see it snap into place.



3. Using the drop-down menu in the **Place at** block, select a dandelion (yellow flower).



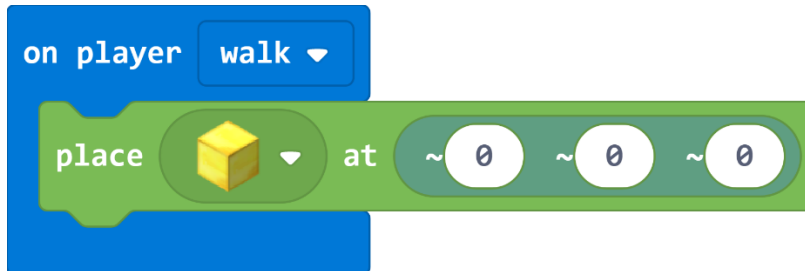
Now, wherever you walk in the game, you will leave a trail of dandelions! Try it out in the game by using the 'W' key on the keyboard to have your Player walk forward in a Minecraft world. Then, look behind you – you should see a trail of flowers!



Teacher Note: We will be exploring the coordinate system in more detail in the next lesson. For now, it's enough to know that the 3 coordinates (X, Y, Z) represent different directions in the Minecraft game:

- X coordinate – East / West
- Y coordinate – Up / Down
- Z coordinate – North / South

Notice that we used the coordinates (~0 ~0 ~0) in our Place block. ~0 in the middle coordinate represents the relative coordinate for ground level. Suppose we wanted to leave a golden path behind you, so that a “yellow brick road” were created wherever you walked? We can do that with the same Place at block, only this time we’ll leave (of course) solid gold blocks.

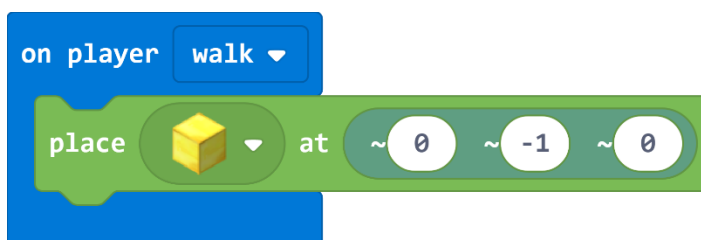


Watch what happens:



The right idea, but not exactly what we are going for. We are actually leaving a gold wall behind us, which is rather inconvenient. How might we sink those blocks into the ground so that they form a yellow brick road?

Let’s modify the Y coordinate by subtracting one so that the bottoms of the bricks are one level down.

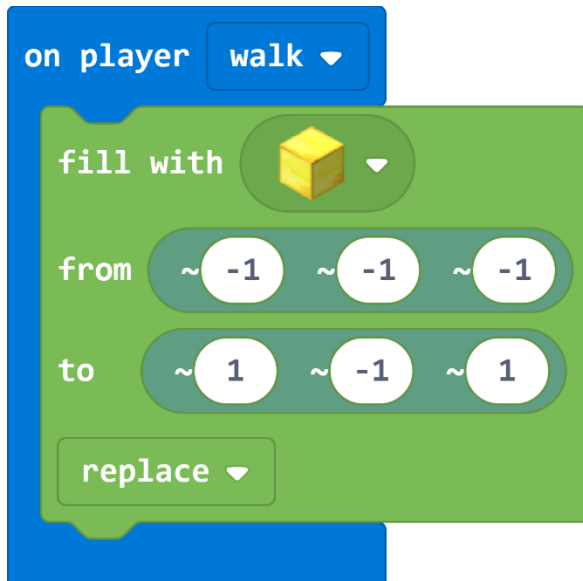


Now, let’s see what happens:



Much better! As an extension, try making it a proper yellow brick road, at least three bricks wide, so you and your friends can walk side by side. Hint: use a different block from the Blocks menu!

Solution:



### Activity: Sing a Song of Sixpence

*"Sing a song of sixpence,*

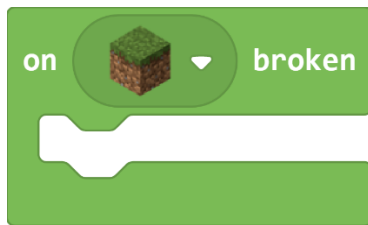


*A pocket full of rye.  
Four and twenty blackbirds,  
Baked in a pie.” – [English Nursery Rhyme](#)*

In this activity, students will take inspiration from this old English nursery rhyme to recreate this in Minecraft. But we shall use parrots instead of blackbirds, and cake instead of a pie!

Steps:

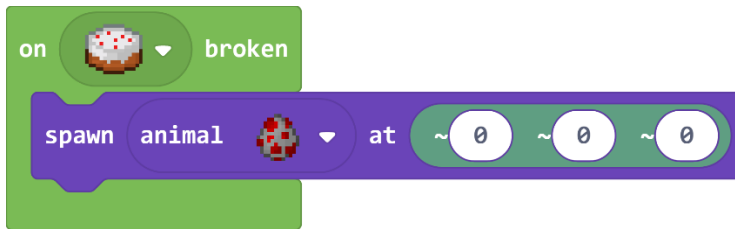
1. From the **Blocks** Toolbox drawer, drag the **On broken** block onto the coding Workspace.  
This will be our event handler.



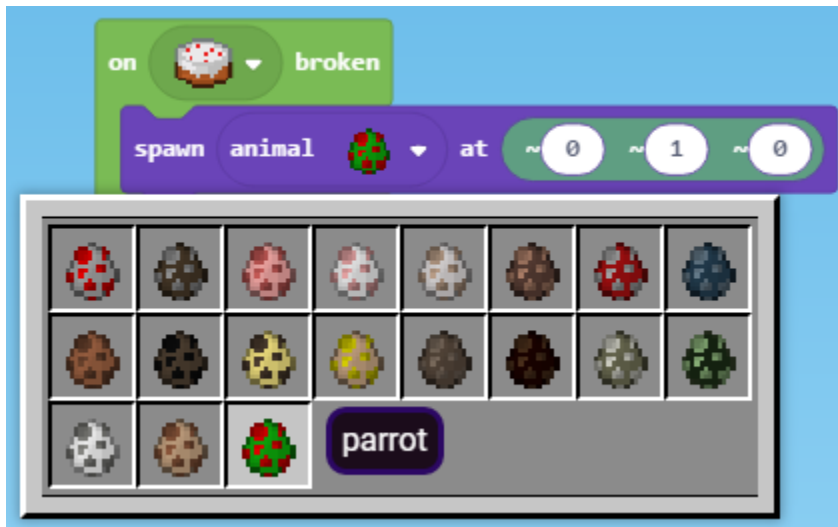
2. Using the drop-down menu, select the 'cake' item



3. From the **Mobs** Toolbox drawer, drag a **Spawn animal** block under the **On broken** block until you hear it snap into place.

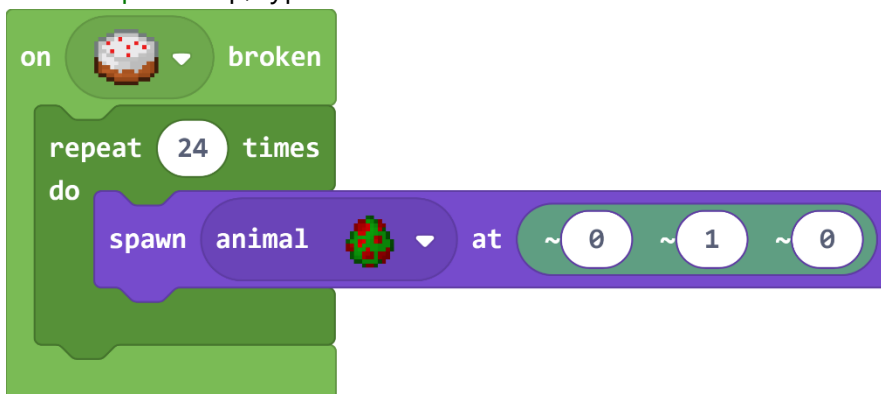


4. Using the drop-down menu in the **Spawn animal** block, select a Parrot
5. We want to spawn the parrots above our head, so in the **Spawn animal** block, change the Y coordinate to 1



This will just spawn 1 parrot above our head, so let's use a **Repeat** loop to spawn 24 parrots.

6. From the **Loops** Toolbox drawer, drag a **Repeat** loop under the **On broken** block, and around our **Spawn animal** block.
7. In the **Repeat** loop, type in the number 24



To run this in the game, add a cake to your Player inventory (press 'E' to open your inventory), place a cake on the ground by selecting it in your toolbar and right-clicking somewhere on the

ground. Then hit it using the left mouse button to destroy it – you should see a flock of parrots appear!



Shared Program: [https://makecode.com/\\_0ji3UvTDg4Ds](https://makecode.com/_0ji3UvTDg4Ds)

### **Activity: Last Stand at the Alamo**

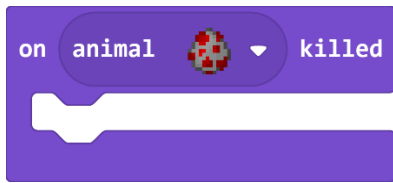
In this activity, we will recreate the experience of being overrun by hordes of zombies. We will also see the effect of exponential growth: every time you kill a zombie, two more spring up in its place. What does that feel like? Let's find out!

To start out with, make sure you are in a safe and/or easily defensible location. Jungle trees work well, as do castle turrets and even a fenced-in corral (where you are on the inside). Alternately, if you are in a flat world, you can just run free, shooting zombies as you go. It's a no-win situation anyway, so you might as well have some fun while you are in it.

The event handler that we will be using is "On monster killed" to trigger new zombies spawning when one is killed.

#### Steps:

1. From the **Mobs** Toolbox drawer, drag an **On killed** block into the coding Workspace

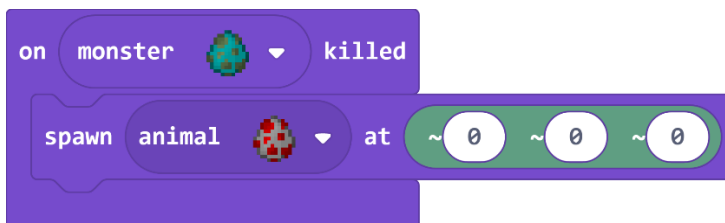


2. From the [Mobs](#) Toolbox drawer, drag a [Monster](#) block into the [On killed](#) block replacing the default animal block
3. Using the drop-down menu in the [Monster](#) block, select the 'zombie' spawn egg

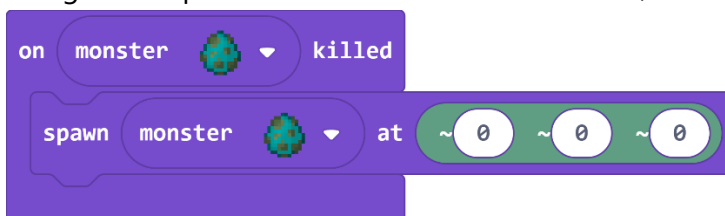


Now, when a zombie is killed, we will spawn two new ones at a random location near the player.

4. From the [Mobs](#) Toolbox drawer, drag a [Spawn animal](#) block under the [On killed](#) event handler block



5. From the [Mobs](#) Toolbox drawer, drag a [Monster](#) block into the [Spawn](#) block
6. Using the drop-down menu in the [Monster](#) block, select the 'zombie' spawn egg



7. We want to spawn 2 zombies for every one we kill, so from the **Loops** Toolbox drawer, drag a **Repeat** loop under the **On kill** block and around our **Spawn** block
8. In the Repeat loop, type in the number 2



Now we need to tell Minecraft where to spawn the Zombies. By default, notice that the relative coordinates are set to your Player's current location (~0 ~0 ~0). Spawning two zombies right on top of you isn't a great idea, so let's make them show up just a little farther away at random locations.

9. From the **Positions** Toolbox drawer, drag a **Pick random position** block and drop into the **Spawn** block replacing the default coordinates

The two sets of coordinates in the **Pick random position** block describe opposite corners of a box within which zombies will spawn at random locations. Let's choose a box 10 blocks around your Player, which should create an area of about 400 square blocks centered on your location.

10. In the **Pick random position** block, type the from coordinates as (~10 ~0 ~10), and the to coordinates as (~-10 ~0 ~-10)



JavaScript:

```

mobs.onMobKilled(mobs.monster(MonsterMob.Zombie),
function () {
  for (let i = 0; i < 2; i++) {

```

```
        mobs.spawn(mobs.monster(MonsterMob.Zombie),
positions.random(
            positions.create(10, 0, 10),
            positions.create(-10, 0, -10)
        ))
    }
})
```

Shared Program: <https://makecode.com/VseXqc2Vjbv9>

To start the fun, open a Minecraft world in Creative mode. Then open your inventory and equip yourself with a weapon (ideally a ranged weapon like a bow), and then give yourself a Zombie spawn egg (you can use the search feature in your inventory).



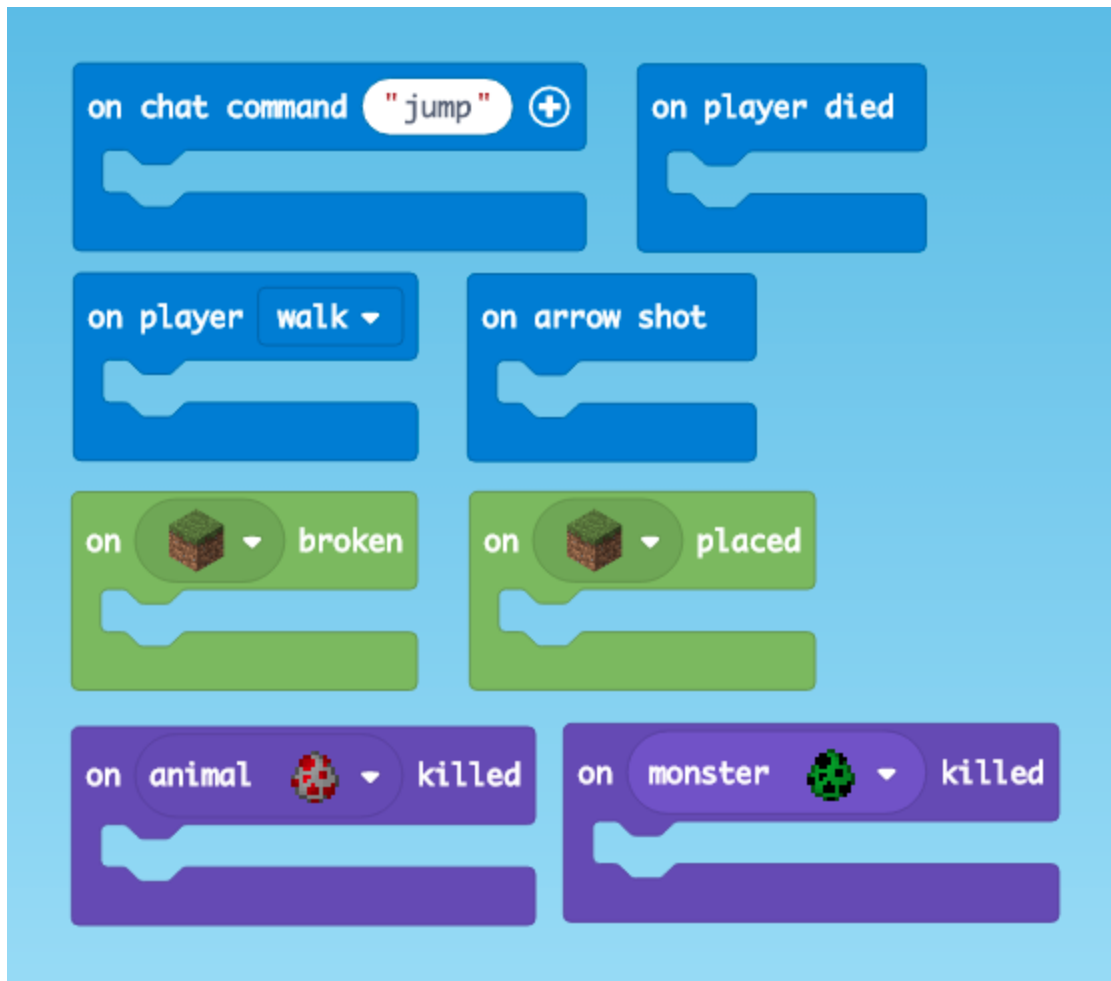
Next, change your game mode to Survival either through your Settings menu, or by typing this command into the chat window: `/gamemode s`. Tip – make sure the Difficulty level of your game is not set to Peaceful – the zombies won’t attack. Finally, get behind a barricade, drop the Zombie spawn egg, and let the fun begin!



## Independent Project

Teacher Note: Open-ended, creative projects are opportunities to apply the concepts and skills students have developed to solve a problem or fill a need. In Minecraft, there are all sorts of problems to solve: How can you keep your cows from running away? How can you create a renewable water source? How can you get back to the surface after falling into a ravine? Projects allow kids to explore problems that are meaningful to them -- and to use MakeCode to design original, automated solutions to many of the problems they face in their everyday Minecraft lives. It is important to give kids opportunities to create original projects to demonstrate that they can apply their MakeCode skills in new ways, and to exercise their creativity.

Remember that we learned in this section that events make things happen during the game. You will need to use event handler blocks to trigger different actions or results. In this project, your challenge is to come up with a simple MakeCode for Minecraft project that uses one or more of the following event handler blocks:



### Event Handler Blocks

- On chat command
- On player died
- On player walk/run/jump/swim/ etc.
- On arrow shot
- On block broken
- On block placed
- On animal killed
- On monster killed

For some of these blocks, you have many choices about the type of block, action, or animal affected.

### Sample Projects:

1. Kaleidoscope Build





In this project, every time you place a cobblestone block, three more are placed symmetrically relative to your location.

X	O	X = block placed O = new blocks
O	O	

## 2. Walk on Water



When you are crossing a lake or a river, make blocks of ice or glass appear under your feet so that you are literally walking on water.

## 3. God of Weather



Create a chat command that makes it rain when it is clear, or turn clear when it is raining.

#### 4. Spite



Create a command that when active, spitefully kills everybody else when you die.

### Minecraft Diary

Compose a diary entry addressing the following:

- What kind of event and event handler did you decide to use?
- What does your program do? Describe how your program works (what is the cause-and-effect)
- Include at least one screenshot of your program working
- Share your project to the web and include the URL here

NOTE: If you decided to improve one of this lesson's activities, please talk about the new code you wrote in addition to what was already provided in the lesson.

### Assessment

	1	2	3	4
Diary	Minecraft Diary entry is missing 4 or more of the required prompts.	Minecraft Diary entry is missing 2 or 3 of the required prompts.	Minecraft Diary entry is missing 1 of the required prompts.	Minecraft Diary addresses all prompts.
Project	Project lacks all of the required elements.	Project: Uses at least one event handler block; OR causes an intentional effect or solves a problem BUT code is ineffective or flawed.	Project: Uses at least one event handler block; OR causes an intentional effect or solves a problem.	Project: Uses at least one event handler block; AND causes an intentional effect or solves a problem.

### CSTA Standards

- CPP.L1:6-05 - Construct a program as a set of step-by-step instructions to be acted out
- CPP.L1:6-06 - Implement problem solutions using a block-based visual programming language