Kids, Code, and Computer Science

beanz

Do You Speak Python?

10 FREE Web Developer Tools

“I had no idea!”

Learn the History of That Game Controller in Your Hand.

The Rail Fence Secret Code

December 2017 $6.00
Publisher’s Note

How old do you think game controllers really are? Old enough to have been used in the 1600s in Rembrandt’s time, as shown on the cover of this issue? Okay, maybe not. But I’m almost certain Rembrandt would have loved video games if he had had electricity and the right tech.

This issue has a few more silly things tossed in with the serious stuff.

For silly serious stuff, there’s an article about Rubber Duck debugging. Turns out talking to a rubber duck is a really good way to fix your code.

We also added a new section about SketchUp, a mostly easy to learn 3D modeling software tool. It’s lots of fun for people not into coding or computer science who want to build things digitally and maybe 3D print. Bonnie, our SketchUp maven, has lots of experience with SketchUp and teaching kids how to use it. You also can find her at 3DVinci.net.

And while lots of people play Minecraft, how many have reached the end of the world? The Nether is a hellish endgame which requires lots of tools and blocks to survive. The End is harder to find but perhaps more interesting. It also gives you a credits screen.

There’s lots to browse through in this issue about Scratch: micro bit, coding, secret codes, and more. I hope you have as much fun reading as we did making this issue! See you in February!

Tim Slavin
Publisher
beanz Magazine

Our Mission

beanz magazine is a bi-monthly online and print magazine about learning to code, computer science, and how we use technology in our daily lives. The magazine includes hard-to-find information, for example, a list of 40+ programming languages for education, coding schools, summer tech camps, and more.

While the magazine is written to help kids ages 8 and older learn about programming and computer science, many readers and subscribers are parents, teachers, and librarians who use the articles to learn alongside their young kids, students, or library patrons. The magazine strives to provide easy to understand how-to information, with a bit of quirky fun. Subscribers support the magazine. There is no advertising to distract readers.

The magazine explores these topics: Basics of programming and where to learn more, Problem solving and collaboration, Mathematical foundations of computing and computer science, Computational thinking, Recognizing and selecting computer devices, and the Community, global, and ethical impacts of technology.
Close your eyes and pretend that you’re playing your favorite video game. Can you picture what’s on the screen? What you’re doing with your hands? Maybe the gameplay feels so natural that you forget you’re using a controller! Tech companies like Nintendo, Sony, and Atari have spent years making video game controllers feel like an extension of your body. Their first controllers, though, weren’t so slick. Let’s hop into a time machine and take a peek at your controller’s ancestors. You can even fetch your controller and see how it compares!

**Spacewar and the PDP-1 (1962)**

Spacewar is considered the first video game. It was designed to show off the power of the new Programmed Data Processor-1 (PDP-1), a computer so massive you’d be lucky to fit it through a doorway. In Spacewar, two blocky spaceships are controlled by eight switches which can be flicked up and down. The switches are lined up in a row and built into the computer itself. The two players had to stand next to each other and fight for elbow room! In fact, the gameplay was so bad that Spacewar’s creators realized they needed to add a small, detachable device to the setup. The first controller was born.

**Arcades**

In the 1970s and 80s, kids spent their time (and quarters) playing arcade games like Pac-Man, Space Invaders, and Pong. Arcades were giant neon-lit rooms. Each machine looked like a cross between a vending machine and a television. Controllers ranged from joysticks to paddles to big plastic guns, and games were played standing up.

**Atari and Pong (1975)**

Pong was so popular that in 1975 Atari made a controller just for Pong: a box with two knobs. Each knob could be twisted to move a paddle up or down. While this worked well for Pong, it became awkward when people tried to play other games. To solve the problem, Atari added a joystick to their controller in 1977. Still, the “Atari Space Tele-Games Pong System” was the first successful “at-home” controller, and it’s the reason you play video games at home today!

**Nintendo Entertainment System (NES) (1990)**

Inspired by miniature calculators, Nintendo wanted to create a device that was small, portable, and cheap. So small and cheap, in fact, that a joystick couldn’t fit on it. To solve this problem Nintendo invented the ‘d-pad’ — a cross-shaped button that could move a character forwards, backwards, or sideways. Now, d-pads are everywhere. Does your controller have one?

**Sony PlayStation Dual-Shock Controller (1998)**

By 1995, controllers looked like spaceships: sleek, aerodynamic, lots of buttons. The dual-shock controller was special because it had two joysticks and a d-pad. So many input possibilities! A player could control their character with one joystick and the camera with the other. The controller also had two ‘rumble-packs’. These were small motors in the sides of the controller that vibrated in response to on-screen events: earthquakes, explosions, etc. Chances are your controller looks a bit like Sony’s dual-shock. What’s different? What’s the same?

**The Nintendo Wii (2006)**

The Wii looked simpler than older models. It was a white rectangle instead of a cool spaceship. But the Wii changed the game by introducing motion sensing. Instead of just sitting around and clicking buttons, or flicking switches, or pulling joysticks, players could swing, punch, throw, and do all kinds of cool movements.

**XBOX Kinect (2010)**

In a sense, the Kinect is the end of the controller. Instead of holding a physical device, players set up a sensor on a flat surface. The sensor detects all their movements — no clicking required! And while the Kinect allowed developers to create a new range of games with full-body motion, playing with the Kinect isn’t as slick as a good ol’ fashioned “spaceship” controller. Controllers have evolved a lot in the last 50 years. And with trends like artificial intelligence, virtual reality, and mobile tech, controllers are going to change even more. What do you think video game controllers will look like in the future?

**ACTIVITY:** Using paper and a pencil, or your favorite sketch app, draw the video game controller of the future.
“Do You Speak Python?”

What number am I thinking of?

Your friend asks. Now, you can play this classic game on a computer, using Python to program it. Python is a simple, flexible programming language with minimal syntax. In other words, you don’t have to worry about forgetting brackets and semicolons. Instead, Python uses indents to decide which lines of code to execute. An indent is created using 4 spaces. Instead, don’t have to worry about forgetting minimal syntax. In other words, you can use a double equals sign (==) and not a single equals sign (=).

Set Up
1. Navigate to www.replit.it
2. Select 'Python3' from the language box (not Python; Python3).
3. Write the following code in the left-hand editor

<table>
<thead>
<tr>
<th>Coding</th>
<th>left-hand editor</th>
</tr>
</thead>
</table>
| ```python | ```
| `@import random` | |
| `@tries = 7` | |
| `@if @tries > 0: @while @tries > 0: @if @computer_guess == @player_guess: @print(f'Congratulations! You win!') @elif @computer_guess < @player_guess: @print('Too low. Try again.') @else: @print('Too high. Try again.') @tries -= 1` | |

Breakdown
LINES 1-2
If you want a pizza, it’s easier to call a pizza expert than to make a pizza yourself. Using a module is like calling an expert. First, you import the module (line 1). Next you call the module and request something, like a pizza, or a random number between 1 and 100 (line 2).

LINES 3-15
In this game, the player has 7 guesses. The 'tries' variable keeps track of how many guesses are left. The MAIN LOOP: LINES 5-15
Each guessing round goes like this:
1. The player’s guess is recorded (line 6).
2. The number of guesses remaining is decreased (line 7).
3. The player’s guess is compared to the computer’s number and the outcome is printed (lines 9-15).

Picture a ‘while loop’ as a car driving around and around a track until it runs out of gas. In our case, the ‘gas’ is the ‘tries’ variable. As long as the player has more than 0 tries, then our car has gas, and the code inside the while loop (lines 6-15) is repeated. Since ‘tries’ starts at 7, the while loop will execute a maximum of 7 times.

NOTE: A code statement is ‘inside a loop’ if the line starts with an indent and comes after the ‘while’ statement. Inside the circular track the car

arrives at a fork in the road: the ‘if-elif-else’ statement. If the player’s guess matches the computer’s number, the car goes to the right (lines 10-11). If the guess is too high the car goes straight (line 13) and if the guess is too low the car goes left (line 15). Only one of these outcomes is possible each round. Then the car drives back to the beginning of the loop and goes through the branch all over again. Each of the three outcomes contains a print statement that notifies the player about their result. Outcome #1 also has an extra line: `quit()`. This is a second method of exiting a while loop — by exiting the entire program. It’s the equivalent of driving our car into a brick wall because it stops all code execution. If the player runs out of guesses the loop stops. The program continues onto lines 16-17, which inform the player that he or she has lost.

Integers VS Strings
Computers store information in different ways. Two common information types are strings and integers. Strings store text characters. Integers — shortened as int— store whole numbers, like 2, or 27, or 2000. Each information type behaves differently in a program.

for example, If you add the numbers 2 and 5 you will get the number 7. If you add the strings ‘2’ and ‘5’ you will get the string ‘25’. You can convert between the two information types using the functions str and int. You can see these in action in lines 6 and 17.

How to Win
Winning the game is all about eliminating possibilities as quickly as you can. At the beginning, all you know is that the computer’s number is between 1 and 100. Therefore you have 100 possibilities.

Let’s say you guess 80 on your first round. If the computer replies that 80 is too low, you know that the computer’s number is between 81 and 100. This leaves 20 possibilities. You could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!

We call this technique ‘Binary Search’. It’s one of the most important search techniques in computer science because it’s really fast. Without binary search, you could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!

How to Win
Winning the game is all about eliminating possibilities as quickly as you can. At the beginning, all you know is that the computer’s number is between 1 and 100. Therefore you have 100 possibilities.

Let’s say you guess 80 on your first round. If the computer replies that 80 is too low, you know that the computer’s number is between 81 and 100. This leaves 20 possibilities. You could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!

We call this technique ‘Binary Search’. It’s one of the most important search techniques in computer science because it’s really fast. Without binary search, you could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!

Example:
```python
@import random
@tries = 7
@while @tries > 0:
  @if @tries > 0:
    @while @tries > 0:
      @if @computer_guess == @player_guess:
        @print(f'Congratulations! You win!')
      @elif @computer_guess < @player_guess:
        @print('Too low. Try again.')
      @else:
        @print('Too high. Try again.')
      @tries -= 1
  @print('You have no more guesses. Try again.')
```

Play the Game!
Now that you’ve mastered the code, try playing a few rounds against the the computer. Can you come up with a strategy that lets you win every time?

How to Win
Winning the game is all about eliminating possibilities as quickly as you can. At the beginning, all you know is that the computer’s number is between 1 and 100. Therefore you have 100 possibilities. Let’s say you guess 80 on your first round. If the computer replies that 80 is too low, you know that the computer’s number is between 81 and 100. This leaves 20 possibilities. You could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!

We call this technique ‘Binary Search’. It’s one of the most important search techniques in computer science because it’s really fast. Without binary search, you could look at a maximum of 100 numbers to find the right one. But with binary search, you only have to look at a maximum of 7. That’s over 10 times faster!
Round Corners in SketchUp

Creating a Box with Round Corners

SketchUp is a free program for 3D modeling. Most people use it for architecture, interior design, city planning—things like that. But it can be used to model anything in 3D, from something as large as a shopping mall to something as small as a leaf or even a molecule.

SketchUp has a downloadable version called SketchUp Make, which you can get at www.sketchup.com. But they also have a web-based version. It’s called my.sketchup, and to use it just go to www.my.sketchup.com.

This is how to make a box with round corners in my.sketchup. After you go to my.sketchup.com, click “LAUNCH,” and run through all of the introductory messages, here’s what you should see: a man (whose name is Josh) standing on the ground, with three axes around him. The red and green axes are on the ground, the blue axis shoots up from the ground. It’s nice to have Josh there to give a sense of size, but he’s really in the way. On the left side of the screen are the drawing and editing tools. Click the Eraser, or press the E key. Click anywhere on Josh to erase him.

The Rectangle tool is what we need to start the box. Some of the drawing tools have little arrows on their icons—this means the tool is part of a tool group. If you click on that tool, all the tools in the group will fly out. You can either click the Rectangle icon, then Rectangle again, or just press the R key.

If you’re ever unsure about what to do while using a tool, look at the Status Bar in the lower left corner. For a rectangle, the first step is to “Select first corner.”

To start the rectangle, click the origin—the point where all three axes meet. (The rectangle can really be anywhere, but it’s a good habit to always start at the origin.)

As soon as you complete the rectangle, a face is filled in. SketchUp always fills in a face when there is a chain of closed edges.

Round corners are done with arcs. SketchUp has a few different types of arcs, and the one we want is the second in the fly-out group: 2 Point Arc. This tool has the shortcut A.

Rounding is done with arcs. SketchUp has a downloadable version called SketchUp Make, which you can get at www.sketchup.com. But they also have a web-based version. It’s called my.sketchup, and to use it just go to www.my.sketchup.com.

After clicking the origin, don’t keep your mouse button pressed. It’s tempting to want to drag the mouse to the opposite corner, but dragging is NOT what you should do in SketchUp. Instead move your mouse to where you want the second corner, and click.

As soon as you complete the rectangle, a face is filled in. SketchUp always fills in a face when there is a chain of closed edges.

Instead of one click to complete the arc, DOUBLE-CLICK instead. This creates the arc and also fillets (shaves off) the sharp corner.

Now that you know how to do in SketchUp. Instead move your mouse to where you want the second corner, and click.

Now we want to round the corners of this rectangle. Use the Orbit tool, and click and drag (dragging is OK for this), so that you’re looking almost straight down at the rectangle. If you have a three-button, scroll-wheel mouse (which every SketchUp user should have), you don’t need to click Orbit. Just press and hold the middle button, even if that middle button is a wheel, and drag to orbit.

For the other three arcs, we need to know where to start so that all four arcs will be the same exact size. So hover (don’t click) over this endpoint:

Now do the same thing to create this arc: find the place on the other edge that makes the arc preview magenta and double-click to complete (and fill) the second arc.

Now that you know how to create identical arcs, there’s an easier way to do it. Place your mouse inside another sharp corner, and double-click. This repeats the same rounding. Double-click for that last corner, and now all four arcs are done.

Now we can pull up the box. But first, orbit to a view like this, which will make the box easier to pull up.

To paint the rectangle, look to the right side of the screen and click the icon that looks like a die. This opens the collections of colors and materials.

In the Materials window that opens, click the Browse (magnifying glass) icon.

In the list of collections, scroll down to find “Wood.” (Or open whatever collection you like.) Click on the wood (or other) material, then click the rectangle to paint it.

To make this box go from 2D to 3D, click the Push/Pull flyout, then the Push/Pull tool. Or press the P shortcut.

Click the rectangle, move the mouse up (don’t keep the button pressed), then click again when your box is the height you want.

We have a box with round vertical corners. But what if the top edge also needs to be rounded? It’s not easy to place arcs when there’s no good face to draw one on. So we’ll need to use a cutout trick.

Draw a rectangle on one of the flat faces. The rectangle should share an edge with the top of the box. The rectangle needs space to the right and left within the flat face.

Use Push/Pull to push in this rectangle.

CONTINUED, NEXT PAGE
BY ERIN WINNICK

Scan the World

3D models and 3D printing can have uses beyond just creating small action figures for your home. One example is the Scan the World project. Begun in 2014, Scan the World uses these technologies to help preserve cultural artifacts, and also to give kids in classrooms a physical connection to the things they are learning. Scan the World gives people the chance to experience tangible representations of artifacts through the use of 3D printing. For people who cannot travel, Scan the World allows users to go online and search for the object they want to see. 3D printing then allows for these shapes to be built up layer by layer, creating a small model of the object.

Using 3D scanning technology and the help of people around the world just like you, 3D models of statues and artifacts are being created for this program. 3D scanning allows people to turn objects into 3D models that can be replicated through 3D printing, or examined on the computer. To date, there have been over 7,500 models created that you can find online. Every object added to the Scan the World library comes from scanned data composed of a series of around 50 overlapping photographs. Most of the photos are taken from a distance away so they encompass the entire item or sculpture, while some are close-ups showing the small details.

Objects in the Scan the World library range from the Statue of Liberty to the Rosetta Stone to the large heads on Easter Island. Scan the World has become a community-built platform and encourages people to get involved in different ways.

You can be a part of this mission to bring history into the computer and keep the art and artifacts alive forever. This project is hosted on the website MyMiniFactory, aimed at offering high quality 3D models. One of the easiest ways to participate is to create a scan of a sculpture using photogrammetry, and sending the photos to those running Scan the World. Photogrammetry is a process that uses a series of around 50 overlapping photographs. Most of the photos are taken from a distance away so they encompass the entire item or sculpture, while some are close-ups showing the small details.

The data that is collected for the objects is processed and managed by users and the Scan the World professionals alike, allowing for the formation of an online digital museum. We all can enjoy these models for free! Scan the World makes our important historical items more accessible to everyone and gives us all a chance to enjoy, appreciate, and learn from them. Visit their website and download your favorite models to experiment with on your computer. Or print them with your own 3D printer or through a 3D printing service.

CONTINUED FROM PREVIOUS PAGE

Use 2 Point Arc to draw another magenta arc, this time on the vertical face made by the cutout.

Now it’s time for the Follow Me tool. This tool takes any 2D face, and “drives” it along a path you select. First, select the path. Click the Select tool (arrow), or press the Spacebar shortcut.

Double-click on the top face of the box. This selects not only the face, but all of its edges as well. We don’t need all of these selected, so let’s unselect some. Press and hold the Shift key, and click the face to unselect it. With Shift still pressed, click the four edges that make up the top of the cutout. Leave only the edges that go around the front, sides, and back selected.

With all of those edges still selected, click the Follow Me tool, in the Push/Pull flyout.

Click the small face in the corner where you drew your arc.

Pretty cool! For more on SketchUp, check out the link on the back cover of this issue.
In the last two issues, we’ve been exploring ciphers that use simple substitution. The first swapped letters of the alphabet around. The next used ‘pigpens’ to designate each letter, then used them to deliver an encrypted code. This issue, we’re going to be looking at a code that doesn’t simply replace letters with other symbols.

Let’s learn Rail Fence Cipher, a very interesting way of delivering messages. We’ll still be using letters in our code, but they will be so scrambled up that anyone who doesn’t know the code won’t know what it says. Even better, unlike simple substitution codes, it’s a little trickier to crack!

How do you make a code with the Rail Fence Cipher? To start, think of a message you want to send. For this example, we want to send the message “HELLO WORLD”.

Then, decide on the amount of ‘rails’ you want to use. For this example, we’re going to use three rails. However many you choose, draw them out like this:

Now, simply write your code out on the rails, so that it ‘zig zags’ up and down on these lines. This is why this code is also known as the ‘zig zag cipher’! The first five letters of our message, ‘HELLO’, will go on the rails like this:

You can see that the word goes down, then up. You can choose to go up first, but whatever route you pick, make sure you remember it, as it’s vital for your recipient to decode your message! Once you’ve gotten started it’s a case of continuing the zig zag until our message is entirely on the rails:

So now we have our message on the rails; but how do we make a cipher out of this? Even if we sent this code as-is, it’s easy to decipher by just reading the message as it goes up and down. That’s why we don’t send it as-is; instead, we’ll write down the letters as they appear on each rail. We’ll also make sure to put a space when we change rails, so our recipient knows to do the same when they come to decipher this message.

So, let’s get started on “HELLO WORLD”. If we look at the top line of our finished rail cipher, we can see it reads “HOL”. The second line reads “ELWRD”, and the third rail reads “LO”:

You can see that the word goes down, then up. You can choose to go up first, but whatever route you pick, make sure you remember it, as it’s vital for your recipient to decode your message! Once you’ve gotten started it’s a case of continuing the zig zag until our message is entirely on the rails:

So, when we go to encrypt “HELLO WORLD” via a rail cipher, it will become “HOL ELWRD LO”!

Then, “ELWRD” is on the second rail, so they’ll write that on the second rail’s spots:

Finally, the third rail has “LO” written onto it.

Your friend can now read your code via the zigzag method.

Now it’s your turn. Let’s say you and a friend have agreed to use a 3-rail system when transmitting codes. You want to send them the message “WHERE SHALL WE MEET?”. After you placed the message onto the rails, how does the encrypted message read?

Similarly, after you’ve given your friend the code, they send a message back that’s to be decrypted on three rails. The message reads: “BEEFS YHCNMATRITIAEX”. What is your friend trying to tell you? Draw out the three rails and put the message onto them starting from the top rail, remembering to leave enough space on each rail for the others!

If you’re interested in this kind of cipher, there’s a lot out there to read. This specific kind of cipher is called a ‘transposition cipher’ because we’re taking a message and transposing the letters around in a logical way so our target can ‘unravel’ the message on their side.
Have you ever wanted to make a videogame, but you’re more comfortable as a writer than an artist? Maybe you want to make stories you’ve written come to life? Or maybe you just want an easy way to get started with games without having to be an experienced programmer? Then you might want to try learning how to write interactive fiction.

Interactive fiction has been a part of videogames since the beginning of computers. Back in 1976, Will Crowther created the very first “interactive fiction” game, Colossal Cave Adventure, with every scene described in text. The player typed in a description of what he or she wanted to do next.

Colossal Cave Adventure was the prototype for text adventure games. These games were most popular back in the ’70s and ’80s, but they’re still being made today! A program called Inform7 is a popular way to design these kinds of interactive stories. One of my personal favorites is a game called Photopia, featuring a short and easy to use technology: HTML, CSS, & JavaScript. These are already the building blocks of every website.

Text adventures aren’t the only kind of interactive fiction games that people make and play. There are also visual novels, which are much more focused on art. Visual novels tend to give the player fewer choices in the story progression.

A great example of a visual novel is the game Long Live The Queen in which the player is a princess trying to survive subterfuge and court intrigue long enough to claim the throne. It was written with a framework called “Ren’Py”, allowing users to create games in the Python programming language.

Another easy to use, yet flexible system for making different styles of interactive fiction is called Twine. Twine is built around the idea of links and passages. You write passages that can be scenes, or descriptions of things, or locations, or even menus for managing your character’s inventory. In Twine, you can tell nearly any type of story. It’s centered around text, but you can still include images and play around with the formatting and appearance of your story.

Twine is built off of accessible, and easy to use technology: HTML, CSS, & JavaScript. These are already the building blocks of every website. To start with Twine, go through the main website: http://twillery.org/. Just click on the link that says “use it online”. After a short explanation of what Twine is and a link to their wiki, click on the +Story button to start a new project. Now you’re ready for the two most fundamental concepts of Twine: the passage and the link.

A passage is displayed on the screen. In a Twine game, you’re always in a passage. But passages can mean a lot of things. They can represent scenes or chapters in the story, dialogues between characters, or places the player can visit. If you want multiple rooms in a house, those might be different passages. If you want different choices in conversations, you can have different passages correspond to different dialogue outcomes. Even complicated ideas like combat or inventory systems will be encoded as separate passages.

How do you connect passages? With links. You can make links inside a passage with the double brackets syntax [[NameOfPassage]], like in the following image.

With links and passages, you can create complicated interactive fiction stories. You can have conversations and adventures, exploration and puzzles. But that’s not all! Twine can do so much more than that. There’s an entire programming environment for you to use as you learn Twine. At https://clarissalittler.github.io/interactive-fiction/TwineDemo.html there’s a puzzle game in which you can accidentally blow yourself up if you enter the wrong code enough times. There’s a horror-ish story where you have to figure out how to avoid being possessed by an alien intelligence. And an adventure game in which you need to escape a cave. You can download this game from github at https://github.com/clarissalittler/interactive-fiction and load it into Twine yourself. From there you can look at the code and learn more about how to do simple programming in Twine.
In past issues, we’ve done some basic and not-so-basic Scratch programming. Now we’re ready to make a “multimodal” game that shifts between two different modes of play: an old fashioned side scroller and an arcade-style shooter game.

First, we need to change which sprite the character uses when the backdrop changes. I have a small example here at https://scratch.mit.edu/projects/170371110 that shows my method for switching which sprite is the “active” one. Click the initial text and then move to a scene with a little beetle sprite.

You can move the beetle all the way to the right, and then the scene changes to a car that you can move all the way to the left to go back to the scene with the beetle. These aren’t just different costumes for the same sprite, we’re actively changing which sprite is getting input.

In code, we do this by giving each sprite a variable that controls whether it is the active sprite that should be receiving user input. Let’s review simple platforming, and we’ll include a health meter made up of three hearts. If you get touched by enemies three times, you die, and the level starts over!

• Choose a sprite for the player to control.
• Give the sprite variables for health, x-velocity (the rate at which the x-coordinate is changing), and y-velocity (the rate at which the y-coordinate is changing).
• Write code to handle the controls (you should be able to move left and right as well as jump).
• Write a main loop that runs when the level starts.
• Design the level (you can either draw platforms as a consistent color in the backdrop or you can make sprites that correspond to platforms and then clone or duplicate them to fill the level). Here’s an example of what it might look like by quickly painting the background.

I made the platforms with two to avoid getting stuck if we jump against the bottom of the platform because touching green counts as “the ground.”

A simple loop might look like:

```
Ghosts will fly across the screen shooting at the player randomly. Stop their projectiles by shooting them before they hit you. Aim by moving the mouse, fire by clicking.

The idea of depth is that every sprite except for the target reticle has a “depth” variable that we call “z” for “z-axis”, which is typically the way depth is described in coordinates. We have a z of 10 for the position of the player and a z of 0 for infinitely far away. We represent depth by shrinking the objects.

If you did the previous Scratch project with the player and AI shooting at each other, you’ve seen clones. Our example is found at https://scratch.mit.edu/projects/169474187/
```

For info on Scratch go to https://scratch.mit.edu/
In past issues, we’ve looked at how to download the game, and how to get the best equipment. But it’s no fun having amazing gear without some way to use it! Try exploring Minecraft’s ‘endgame’. It’s very challenging.

The endgame consists of two realms: the Nether, and the (suitably named) End.

The Nether
The Nether is the easier endgame realm; however, it’s definitely not easy to survive in! It is a hellish landscape with treasures within and unique ores and blocks to be gathered. First, gather obsidian blocks. Technically you need 14 of them, but there’s a way to construct an ‘economical’ version with only 10.

To make obsidian, pour water over a ‘source block’ of lava. Use a bucket of water, or redirect an existing water flow. Obsidian blocks can only be mined by a diamond pickaxe, so bring one with you. When you have 14 obsidian blocks, build a portal in a 4x3 rectangle like this (left).

If you’re building the economy version, you can replace the corners with anything.

To activate the portal, place fire in its center by using flint and steel. Take a block of iron and some flint (usually found in gravel) and combine them like this (left).

With the flint and steel equipped, right click the top of the bottom blocks of the portal to activate it.

Don’t worry about making a portal on the other side; one is already created for you when you enter the Nether. Get ready, then stand within the purple portal. Wait five seconds, and you’ll be teleported into the Nether.

The Nether contains a lot of nasty creatures, like zombie pigmen, flying ghasts, and fire elementals. But it’s the only place to find soul sand and glowstone. It also contains nether fortresses containing chests of loot, and can even be captured for pre-built homes!

The End
The End is a much more suitable term for the ‘endgame’, as completing it will give you a credits screen. Unfortunately, it’s a little harder to find than the Nether.

First, make an Eye of Ender by combining Blaze Powder and an Ender Pearl. Blaze rods can be found in the Nether by killing a Blaze. They look like this (left).

Once you’ve obtained a Blaze Rod, put it in your crafting area to make two blaze powders.

Ender pearls are obtained by slaying an Enderman or finding one in a Nether stronghold chest. Endermen typically spawn in the regular world, but they’re quite rare and hate being looked at directly! They look like this (left).

Once you’ve obtained an Ender Pearl, combine it with one blaze powder (right).

With the Eye equipped, right click to throw it. The Eye of Ender will then float in the direction of the portal to The End. If it also floats upward, it means there’s a long distance between yourself and The End. It also has a chance to break once it falls, so make sure you create extras.

The nearest portal to The End may be many miles away. You may need to place markers and come back later. Once you get close, the eye will start going downward into the ground. Dig where the Eye is travelling. If you dig deep enough, you’ll find a stronghold. Bring plenty of supplies and at least 12 Eyes of Ender. Eventually you’ll find a room with a portal (right).

You’ll see that there are eyes in some of the slots around the portal, but not all. Fill in the rest of the portal and it will open up.

The End is very difficult, and requires fighting a lot of Endermen and an Ender Dragon. You can’t return until you’ve won the fight, so don’t enter immediately. Check the link on the back cover for an in-depth guide on fighting the dragon, so you can be fully ready for what’s waiting for you.

Good luck!
Kotlin: A New(ish) Superhero

Kotlin is about six years old, young by programming language standards, but it’s one of the only languages officially supported for Android development! It’s used by companies like Pinterest and Coursera for their Android apps, impressive for a young language.

Kotlin was developed by JetBrains, the company behind IntelliJ, which is one of the big development tools for Java. Kotlin was designed to be a language that can function as a drop-in replacement for Java. It is meant to address some of the complaints people have about Java, like how infamously verbose it is.

Kotlin is based on the same framework as Java: the Java Virtual Machine (JVM) and all the library code that comes with it. A virtual machine is like an emulator for an old video game. Instead of having a program that pretends to be a Super Nintendo running on your computer, it’s a program that pretends to be an imaginary computer whose only job is to run something called JVM bytecode.

The way Java solved the problem was to shift the burden of portability from the working programmer to the Java implementation itself. The programmer is isolated from the working programmer systems and hardware and, instead, types that are checked to determine whether a program is “error free” and should be allowed to run?

Languages like Python, Ruby, or JavaScript that are compiled to JVM bytecode, are “untyped”. I think typed and untyped languages. This is why Kotlin and Python look different at first glance: we’re having to name all these types. Sometimes people will say “statically typed” while I say “typed”, or “dynamically typed” while I say “untyped”. I think typed and untyped get at the heart of it better: are there types that are checked to determine whether Kotlin and Python look different at first glance: we’re having to name all these types. Sometimes people will say “statically typed” while I say “typed”, or “dynamically typed” while I say “untyped”. I think typed and untyped get at the heart of it better: are there types that are checked to determine whether a program is “error free” and should be allowed to run?

So when we’re declaring our variable that stores the length of the list. The last unusual bit of syntax is a, b -> a < b). If you know Ruby you might think this looks a little bit like a block and you’d be absolutely right! You can declare small functions like you might with Python’s lambda keyword or Ruby’s block syntax this way.

If Kotlin look interesting to you, or you’re interested in something other than Java for Android programming, then check out the link on the back cover of this issue for additional reading, tutorials, and more!

```kotlin
fun <T> bubbleSort (l : MutableList<T>, c : (T,T) -> Boolean) : MutableList<T> { val length = l.size var isSorted = false while (!isSorted) { isSorted = true for(i in 0..length-2) { if(c(l[i],l[i+1])){ var tmp = l[i+1] l[i+1] = l[i] l[i] = tmp } isSorted = false } } return l }

fun main(args : Array<String>) { val testList = mutableListOf<Int>(4,2,3,1)
    val length = testList.size
    var isSorted = false
    while (!isSorted) {
        isSorted = true
        for(i in 0..length-2) {
            if(testList[i] > testList[i+1]) {
                var tmp = testList[i+1]
                testList[i+1] = testList[i]
                testList[i] = tmp
            } else {
                isSorted = false
            }
        }
    }
    println(testList)
}
```

This might look a little intimidating, so let’s break it down: we want to sort lists. In Kotlin, lists that you can modify are mutable lists which have a type of MutableList. Our sorting function has two arguments: the list to sort and a function that defines what “greater than” means for sorting. If you wanted to, say, sort a list of numbers from low to high, then you’d want to compare them by the normal > operator.

If you’re not familiar with Java, you might not know what this `<T>` business is. In Python, for example, you can just write your bubble sort once and it can work on a list that has anything in it. That’s because Python is untyped! In Kotlin, though, you need to use a generic parameter to name the type that the list is going to hold. It doesn’t have to be `<T>`. It could be `<FatDog>` for all Kotlin cares. It just needs to be a name!

There are two different ways to create variables in Kotlin: var and val. var is for things that you want to change over time. val is for things that shouldn’t change, like our variable that stores the length of the list.

```
val isSorted = False
```

The programmer is isolated from the working programmer systems and hardware and, instead, writes code for the virtual machine. It was the job of Sun Microsystems, and now Oracle, to make programs that could run the virtual machine on every platform.

This portability and history is why there are a number of languages that are compiled to JVM bytecode, such as Scala, Clojure, and now Kotlin.

Here’s an example of a Kotlin program that implements a bubble sort. You can provide the function used for comparison as an argument.

```
fun <T> bubbleSort (l :.MutableList<T>, c : (T,T) -> Boolean) : MutableList<T> { val length = l.size var isSorted = false while (!isSorted) { isSorted = true for(i in 0..length-2) { if(c(l[i],l[i+1])){ var tmp = l[i+1] l[i+1] = l[i] l[i] = tmp } isSorted = false } } return l }
```

```
fun main(args : Array<String>) { val testList = mutableListOf<Int>(4,2,3,1)
    val length = testList.size
    var isSorted = false
    while (!isSorted) {
        isSorted = true
        for(i in 0..length-2) {
            if(testList[i] > testList[i+1]) {
                var tmp = testList[i+1]
                testList[i+1] = testList[i]
                testList[i] = tmp
            } else {
                isSorted = false
            }
        }
    }
    println(testList)
}
```
1. **Link Sleuth**

Link Sleuth is a Windows application that lets you check all of the links on your site to make sure there are no broken links. You can customize the analysis and it generates a report in HTML for you.

http://home.snafu.de/tilman/xenulink.html

2. **HT Track**

HT Track is a great tool for copying websites. You point it to the root of the site you want to copy and it downloads all of the HTML, CSS, scripts, images, etc. This is great if you have a client who has a flat HTML site and wants you to put it into a Content Management System for them. It is also good if you are taking over a site for a client but you don’t have the source code.

https://www.httrack.com/

3. **Trello**

Trello is a great tool for organizing your work or your processes. You create a board and in the board you have lists. In those lists you add cards. Where I work currently, we have a board for our web projects. We have lists for: potential work, quoted for, quote accepted, work in progress, work completed, and archive. The jobs start on the first list, Potential Work, and they move into each list as they get to the next stage. You can also use it for planning your day. You can use it to plan out where you are going to use your pomodoros for the day and for the rest of the week.

https://trello.com/

4. **Slack**

Slack is a messaging app for teams to communicate about the projects they are working on. It is instant messaging with lots of extra features. Many other programs integrate with Slack, like bug trackers which send notifications to your team chat. Slack is available as a desktop or mobile app and of course on the web.

https://slack.com/

5. **Telerik Converter**

Telerik converter lets you convert code from C# to VB and from VB to C#. You just paste the code you want to convert and it converts it for you.

http://converter.telerik.com/

6. **Built With**

If you want to know what technology has been used to build a certain site, you can use this site to find out.

http://builtwith.com/

7. **Can I Use**

“Can I use” provides up-to-date browser support tables for support of front-end web technologies on desktop and mobile web browsers. It is a very handy tool when you want to check if a certain html or CSS tag will be compatible for the browsers you need to support.

https://caniuse.com/

8. **WebPageTest**

WebPageTest is a free tool which checks the speed and performance of your site. It gives you ratings for different aspects and tells you what you need to do to improve.

http://www.webpagetest.org/

9. **LINQPad**

LINQPad is ideal if you find some code on the internet that you want to quickly test, or if you want to play with the new features in C# 6 without having to set up a windows console/forms app or website.

http://www.linqpad.net/

10. **LoremPixel**

LoremPixel gives you placeholder images to use in your site concepts or examples. It randomly picks an image each time the page loads. You can request the images at your own custom size.

http://lorempixel.com/
In the last two issues we introduced the micro:bit by using it with a block based JavaScript programming language, formerly called PXT. But now we’ll learn how to use Python with the micro:bit to create a 3D light sculpture using LEDs called Neopixels, a brand name created by Adafruit. The formal name for these LEDs is WS2812B. The super bright LEDs are sold in many shapes. From individual LEDs called “pixels”, to rings, squares and super long strings of pixels!

You’ll Need:
- A micro:bit
- Micro USB lead to connect micro:bit to computer
- 3 x Neopixel rings of different sizes (24, 16, 7)
- A Kitronik breakout board, http://www.mcmelectronics.com/product/83-17968 or to make your own, see the end of this article.
- Breadboard
- A soldering iron
- Solder
- Sponge & stand
- 3x Single Core Wires

For info on Micro:bit go to https://microbit.org/resellers

The circuit consists of the neopixels, the micro:bit and the power source for the neopixels, connected with a “common ground”.

Now let’s solder three wires to the largest neopixel ring. Ensure they are different colors and that you stick to these colors! We used Power/VCC = RED, GND/Ground = GREEN, Data (In and Out) = YELLOW.

You’ll Learn:
- Importing libraries of pre-written code
- Using loops to repeat code indefinitely
- Using tuples to store RGB color values
- Creating animations via algorithms
- How to create electronic circuits with soldering irons.

First, solder the neopixel rings together. Set up your soldering area so that you have easy and free access to your soldering iron and everything you need. Turn on the iron and let it warm up. Do not leave it unattended. You will see that each neopixel ring has solder points labelled IN, OUT, V / PWIR and GND. Add a little solder to each of these points. Take your time, wear protective glasses, and solder in a well-ventilated room. Children should be supervised, or parents may wish to tackle the soldering.

Now we need to cut, strip and “tin.” Take a length of each color single core wire, and cut each length into three equal parts. Using your wire strippers, strip about 1cm from the ends of each wire. Using a blob of modelling clay to hold each wire, gently touch the hot soldering iron to the wire and then touch the solder to the wire. After a second it should flow and coat the wire in solder. Repeat for all of the wires.

Now let’s solder three wires to the largest neopixel ring. Ensure they are different colors and that you stick to these colors! We used Power/VCC = RED, GND/Ground = GREEN, Data (In and Out) = YELLOW.

With all of the rings soldered, now connect the wires from the largest ring to the breadboard. Our breadboard has two “rails” along its longest two sides. These are + and -, in other words VCC (+) and GND (-). Connect the VCC and GND from the largest ring to these rails. There remains one wire left to connect on the ring, but you’ll do that later.

Now connect the VCC and GND wires from the battery to these rails. Using the breakout board, connect pin 0 of the micro:bit to a spare hole in the breadboard, but not the rails. Next, take the IN wire
from the largest ring and place it in line with the pin0 wire. Pin0 is now connected to our input pin and can control the neopixels.

Lastly, connect the GND pin on the micro:bit to the (-) rail of the breadboard. This creates a “common ground” reference, and enables all parts of the project to talk to one another. With all of the hardware connections made, go to http://python.microbit.org/editor.html Here you can write the Python code that will control the neopixels. Then download the code compiled into a .hex file. Copy the hex file to the flash drive when connected to the micro:bit which appears as a USB drive when connected to the computer.

Start by importing the micro:bit Python library, which enables you to use the pins of the micro:bit. Next, import the neopixel library so that you can control the neopixel rings with this Python code.

```python
import neopixel
from microbit import *

np = neopixel.NeoPixel(pin0, 47)
```

Using a loop, run the code inside the loop “forever”, and the first line of code in the loop is indented for us. This identifies that the code is to run in the loop. We create a “for” loop, which is a loop that will go round a set number of times. In this case it will go round 12 times, starting at 0 and advancing 2 steps at a time (2 * 12 = 24, the number of LEDs in the largest neopixel ring). Each time the for loop goes round, it will change the color of a single LED in the neopixel ring, 'np[i]' where ‘i’ is an even number (0,2,4,6...12) so the first time round the first led lights up. Then, the loop goes round and we skip an LED and go to the next. This gives us a staggered pixel effect, similar to stripes on a candy cane. To control the color of the LED, use three numbers that represent Red, Green and Blue. These go from 0 to 255, with 0 = LED off, and 255 being full brightness. In the first loop set the red level to 30 and the rest to 0 to get a pleasant red glow. In order to see the LED color change, you need to instruct the LED to “show” the change. Lastly, instruct the code to wait for 0.1 seconds.

```python
for i in range(0,24,2):
    np[i] = (30,0,0)
    np.show()
    sleep(100)
```

Next, create another for loop, for the same neopixel ring (24 LEDs) but this time start at LED 1 (the second LED on the ring) and advance 2 steps each time the loop goes round. This lights up the odd numbered LEDs. This time light them “0,30,0” which is green.

```python
for i in range(1,24,2):
    np[i] = (0,30,0)
    np.show()
    sleep(100)
```

Now moving on to the next neopixel ring, which has 16 LEDs, repeat the same two for loops, but this time alter the colors so that you have yellow “30,30,0” and green “0,0,30”

```python
for i in range(24,40,2):
    np[i] = (30,30,0)
    np.show()
    sleep(100)
for i in range(25,40,2):
    np[i] = (0,30,0)
    np.show()
    sleep(100)
```

The final neopixel ring has just 7 LEDs. Set the even numbered LEDs to white “30,30,30” and the odd numbered to yellow “30,30,0”. Outside of the for loops we create a delay of 1 second using “sleep”.

```python
for i in range(40,47,2):
    np[i] = (30,30,30)
    np.show()
    sleep(100)
for i in range(41,47,2):
    np[i] = (30,30,0)
    np.show()
    sleep(100)
```

When complete, click on “Save” to download a copy to the computer. Click on “Download” to download a compiled hex file to copy to the micro:bit. Plug the micro:bit into your computer and it will appear as a USB flash drive. Copy the downloaded hex file to the micro:bit and it will start flashing the device. After a few seconds the micro:bit will reboot and the LEDs on all of the rings will light up.

Make your own breakout board using M4 countersunk machine bolts, wire, nylon washers, and nuts. You can learn how to make your own micro:bit to breadboard breakout by visiting http://www.makerspace-uk.co.uk/microkit-a-low-cost-breadboarding-solution/

For a deeper dive into this project including how to add animation, visit the link on the back cover of this issue.
Rubber Duck Debugging

Got an impossible coding error that you can’t figure out, no matter how many times you read your code, line after line? How do you solve an unsolvable problem? Talk to a duck. Or a teddy bear. Or your Star Wars Pop! Vinyl doll. Or your favorite American Girl doll. While it’s called Rubber Duck debugging, any inanimate object works.

If you sit your duck down next to your computer and carefully explain what your code is designed to do, then explain line by line how your code does what it’s supposed to do, at some point you will find a solution to your problem. Chances are, you will find the bug. Or find an idea or two how to fix the problem.

While it’s fun to think of inanimate objects as master coders, when humans explain code in detail, out loud, often they see things they don’t notice when simply reading code. The difference between what we think happens and what actually happens becomes clear.

So keep a duck or other favorite object by your computer when you code. While they might be happy to stand around all day, they also can help when you get stuck coding.

I’m here to help!

What’s Inside?

Ever wonder what’s inside an iPhone? Apple Inc has sold 1.2 billion iPhones in 15 different models before their current iPhone 8 and X releases. Bloomberg joined with iFixIt.com, IHS Markit, and Sunny Lin of Simple Mac, to break down every iPhone model. Check out https://www.bloomberg.com/features/apple-iphone-guts/ for all the cool details. Here’s a peek at the iPhone 8:

In the US alone, an estimated 33 million video gamers have at least one disability or another. For example, people with color blindness have a difficult time telling the colors red and green apart, colors often used in video games to identify characters or identify correct and incorrect choices. The “Star Wars Battlefront” game offers players multiple color choices, including one that works for people who are color blind. A British hardware company also makes one handed controllers as well as devices to convert head or leg motion into button presses and directional controls. And a company based in California makes chair attachments and backpacks designed to convert low-frequency sounds into vibrations that can be felt throughout the body, allowing deaf gamers to feel the cues.

—Adapted from the article “Games for People With Disabilities,” economist.com, 9/30/17

—Adapted from the article “Games for People With Disabilities,” economist.com, 9/30/17
STEAM events can be a fun way for schools, community libraries, and families to play with technology and gauge the interest level of their communities, for example, whether or not there is enough interest to set up a makerspace. These ideas work for kids of all ages.

Some of these event ideas require little or no technology. You only need a room, someone to lead the event, and people to work with kids throughout the event. The easiest place to start would be with board games. For older kids, they’ll need access to a phone, tablet, laptop, or desktop computer with web browsers and other software.

Resources mentioned are linked in the online version of this article (see the back cover of this issue).

**Kids Ages 5-8**

For this age group, or anyone new to technology, there are board and card games that are fun to play as they teach the basics of computer science and programming.

*Robot Turtles* teaches players about conditional logic and strategic thinking as they direct others through the game. A *Robot Turtles* game can be played by up to 5 people with someone to keep everyone on track throughout the game.

Other board games include *Code Monkey Island* and *Treasure Island*. Fun card games include *littleCodr* and *Bits and Bytes*.

**Kids Ages 8-12**

Events for kids familiar with the basics of CS and programming can evolve into services and languages that make it easy to learn more about programming. There are offerings from simple to more complicated. Some are available online; others are apps which must be downloaded.

Hosting an event would require that kids bring in phones, tablets, and laptops or have them provided by your library or group. Parents or adults would be needed for every 3-5 kids to help them set up and answer questions. Because these programs are available online, often with free versions, it’s not difficult for adults to learn the program before the event and become comfortable with the service or language.

*Tynker.com, Run Marco!, Erase All Kittens, and The Foos* are examples of online services that teach kids how to program in a fun way. *Hopscotch, Scratch, and Scratch Jr.* are equally fun but require downloading an app. For younger kids, *Move the Turtle* is another app which is easy to master.

In addition, *CodingFarmer* is a board game that lets players learn two ways, with code (Java) and without code. It’s similar to games like *Robot Turtles* but with the ability to learn with Java.

**Kids Ages 12 on up**

Older kids who have some experience with Scratch or similar visual block languages might want events where they can learn text languages used by adults. There are apps which provide interactive consoles, also called playgrounds, to allow you to work out problems in Python, JavaScript, Swift, and other languages. An app called *Codea* lets kids create games using Lua, another professional language.

There are apps with interactive consoles that include tutorials. Your event might spend an hour on a few of the lessons. The kids then will have the ability to continue the tutorials after the event. Kids would need laptops and adults supervising the event would need to be comfortable enough to answer questions.

**Beyond Basic Events**

There are more STEAM tools you could build events around. For example, *Sphero* and *Wonder Workshop’s Dash and Dot* robots mix programming and robots. Kids use a visual block language similar to Scratch to tell the robots what to do.

These events would require prior planning and the use of existing lesson plans (or creation of new ones) to ensure the events succeed. Both of these robots include websites with ideas that can be used for events, for example, creating an obstacle course for the robots to complete.

Another option might be *Bitsbox*, a monthly service that delivers easy to create, adaptable phone apps kids can build. This might work as an ongoing event series to keep kids engaged and provide an evolving set of challenges instead of one-off events.

If you’re interested in equally fun but less technical options for STEAM events, look into *Computer Science Unplugged* materials online, as well as reading Lauren Ipsum and the *Computational Fairytale* books to kids.
Players are artists who create their own reality within the game.

—Shigeru Miyamoto

Thank you for reading this issue of beanz! Check out the links below to read stories from this issue online with links to learn more.

History of Game Controllers 101
https://beanzmag.com/history-game-controllers

Do You Speak Python?
https://beanzmag.com/python-guess-the-number

Creating a Box with Round Corners in Sketchup
https://beanzmag.com/sketchup-build-box-round-corners

Scan The World
https://beanzmag.com/scan-the-world

Sitting on the Fence
https://beanzmag.com/codes-fence-cipher

It’s YOUR Write!
https://beanzmag.com/twine-interactive-fiction-games

Scratch That!
https://beanzmag.com/scratch-multimodal-games

The Nether and The End
https://beanzmag.com/minecraft-end-games

Kotlin: the New(ish) Superhero
https://beanzmag.com/kotlin

10 FREE Web Developer Tools
https://beanzmag.com/10-web-tools

See the Light!
https://beanzmag.com/microbit-light-art

Click and Say “Om”!
https://beanzmag.com/css-zen-garden

Rubber Duck Debugging
https://beanzmag.com/rubber-duck-debugging

Tidbitz
https://beanzmag.com/tidbitz-december-2017

Let’s Make It an Event!
https://beanzmag.com/create-tech-events

http://beanzmag.com/subscribe
Subscribe!