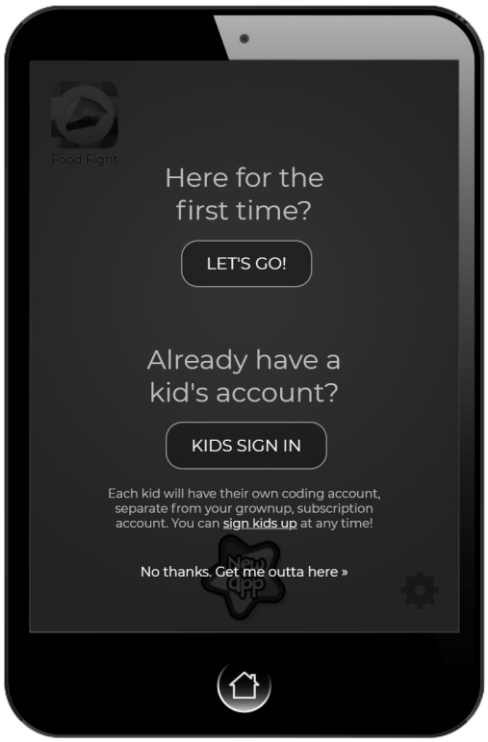
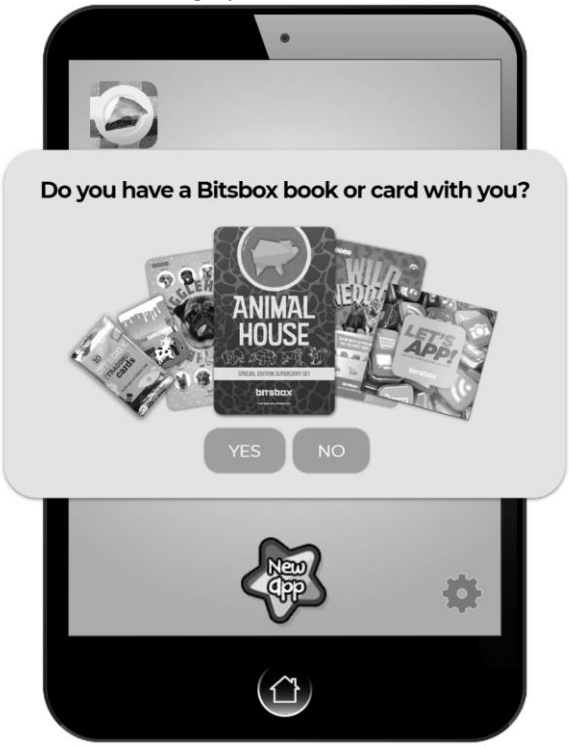
GETTING STARTED

Open up a browser and navigate to:

https://bitsbox.com/code.html

Once there, click the big **GET STARTED** button. That should bring up what looks like a tablet:





If you have your own Bitsbox account, click the **KIDS SIGN IN** button and enter your username and password. If you don't, just click **LET'S GO**. Next, click on the **New App** star, and when prompted for a Bitsbox book or card, click on **NO**. The screen on the tablet will change again; click the **INVENT YOUR OWN APP** button.

RULES

1. The focus of this evening's activities is creating a Bitsbox app; Cubs and Scouts found to be visiting other websites will be warned to return to Bitsbox. After three warnings, the Cub or Scout will be asked to leave the library, and will be given another task for the remainder of the meeting.
2. Cubs and Scouts who engage in disruptive behaviour will be warned to return to working on their apps. After three warnings, the Cub or Scout will be asked to leave the library, and will be given another task for the remainder of the meeting.
3. Any Cub or Scout found to be accessing inappropriate, illicit, or illegal content on their device will be asked to leave the library, and their parents will be contacted to come and collect them. **No warnings will be given.**
4. Cubs and Scouts should work on completing the challenges in the nine stages of this document. As fun as it might be to make an app where e.g. one flushes the faces of different American presidents down a toilet (*this has happened!*), the goal of the evening is to learn several new programming concepts.

STAGE ONE: SETUP

Create a basic app with:

- A character – `stamp()`
- An item or two – `stamp()`
- A background image or colour – `fill()`

Click the  button in the coding window for huge lists of fills, colors, stamps, and more.

See STAGE THREE for a bit more information about how coordinates work in Bitsbox!

EASY

The easiest way to create a **background**:

```
fill('color or name')
```

The easiest way to create a **stamp**:

```
stamp('name')
```

Stamps can have different locations and sizes too (here, **x**, **y**, and **size** would be numbers):

```
stamp('name', x, y, size)
```

To create a **stamp** you can do other things with, make it a **variable** (here, it has the name **a**, but you can give it another name too):

```
a = stamp('name')
```

STAGE TWO: ITEM INTERACTION

Make the item respond to a tap or mouse click.

EASY

Bitsbox has a command called **.tap** which can be attached to a **stamp** to make it react to taps and clicks. To use it, you will need to have given the **stamp** a **variable** name (see STAGE ONE).

A very simple interaction is:

```
a.tap = explode
```

As you can imagine, this will make the stamp explode (mushroom cloud and all) when you tap it. The stamp will then be destroyed, and no longer visible in the program until you run it again.

Click the  button in the coding window and look under the **Commands** tab; there are a few other choices beyond exploding the stamp.

INTERMEDIATE

You can do more with the **.tap** feature if you use a **function**:

```
function tap() {
  a.explode()
}
```

Using **this** means that you can assign the **function** to more than one stamp, and the **function** will work with whichever stamp is interacted with:

```
a = stamp('name')
b = stamp('name')
a.tap = boom
b.tap = boom
function boom() {
  this.explode()
}
```

You can even add multiple commands to functions!

STAGE THREE: CHARACTER MOVEMENT

Make the character move when tapped or clicked on.

It may help you to know, for this section, that the Bitsbox tablet screen is 768 pixels wide and 1024 pixels tall. The centerpoint of the screen is (384,512). Position (0,0) – that's (x,y) – is located in the **upper-left** corner of the screen; x numbers increase as you go **right**, and y numbers increase as you go **down**.

EASY

If you haven't yet started using **functions**, you will have to do so here. If you've just been using **.tap** to make things happen, the easiest way to add sound effects is to turn **tap** from a command into a **function**. For example, if you did this previously:

```
a.tap = explode
```

You can make this into a **function** like so:

```
function boom() {
  a.explode()
}
a.tap = boom
```

The easiest way to make a stamp move in Bitsbox is to use the built-in **move** command. But we need to put this in a function:

```
function zoom() {
  a.move()
}
a.tap = zoom
```

When **move** is used just like that, the stamp will be re-located to a random position somewhere on the tablet screen.

INTERMEDIATE

Both the **move** and **rotate** commands do allow you to send additional information to better control how the **stamp** moves.

If you supply the **move** command with numbers representing x and y coordinates, the stamp will instantly move to those coordinates when tapped:

```
a.move(x,y)
```

You can also give a time value (in milliseconds) if you want the stamp to slide over to its new position:

```
a.move(x,y,time)
```

The **rotate** command can be told how many degrees to rotate the stamp by:

```
a.rotate(number, time)
```

There are other options that can be used here; instead of coordinates, the **move** function can take directions as inputs (e.g. UP, NORTH, RIGHT, LEFT, SOUTH, DOWN). And the **rotate** function can take some of those keywords as inputs as well.

CHALLENGING

You can also make the character move to any point you tapped on the screen:

```
a = stamp('name')
function tap() {
  a.move(x,y,time)
}
```

What's different this time is that instead of supplying numbers to replace **x** and **y** in the **move** command, leave them as letters; you only need to supply a number for the **time** it should take the stamp to move to its new coordinates. In Bitsbox, the **x** and **y** built-in variables are set every time the screen is tapped or clicked; they store the x and y positions of the tap.

There's also an alternative to using **tap**; you can use the **touching** command to only move the stamp while the screen is being touched

```
a = stamp('name')
function touching() {
  a.move(x,y,time)
}
```

But **touching** doesn't work with mouse clicks!

EASY	INTERMEDIATE	CHALLENGING

Another command that can move a stamp is the **rotate** command:
`a.rotate()`
 	
As with **move**, **rotate** used in this way will cause the stamp to rotate (in place) a random number of degrees. If you want a **stamp** to rotate only when tapped, you will again need to use a **function**.
		That's ok, because Bitsbox gives us two other commands that we can use instead: **touch** and **untouch**.
		`a = stamp('name')`
		`function touch() {`
		`  a.move(x,y,1000)`
		`}`
		`function untouch() {`
		`  a.move(a.x,a.y)`
		`}`
		
		We have to do something clever for the **untouch** command. In addition to tracking the x and y coordinates of each touch/click, Bitsbox tracks the x and y coordinates of each **stamp**. To stop the stamp from moving when we release the mouse (or stop touching the screen), we need to tell the stamp to move *to its current x and y position*.

STAGE FOUR: ADDING SOUND EFFECTS

Add some sound effects to item interactions, character movement, or both.

EASY

You will need to work with **functions** again to make this work. Remember the **function** example from STAGE THREE?

```
function boom() {
  a.explode()
}
a.tap = boom
```

First, if you want the function to work with more than just the one **stamp** named **a**, use the built-in **this** variable:

```
function boom() {
  this.explode()
}
a.tap = boom
b.tap = boom
```

Using **this** means that you can assign the **function** to more than one stamp, and the **function** will work with whichever stamp is interacted with.

You can then add a sound effect to the function:

```
function boom() {
  sound('name')
  this.explode()
}
```

Click the  button for a list of sounds.

INTERMEDIATE

If you spend some time exploring the list of sound effects in the  menu, you'll probably notice there are a fair number of similar sound effects. For example, there's the **boom** sound effect for explosions, but there's also **bang** and **kaboom**, **explode**, **exploding**, and **explosion**. (There's also **explosion2**, but it doesn't sound like much.)

So, if we are making a **stamp** explode, is there a way we can have different sound effects each time? Why, yes, there is; we can use the **random** function, along with an **array**. An **array** is just a container; if a **variable** can hold one piece of information (like a **stamp**), an **array** can hold multiple pieces of information. And the **random** function can then choose one of those pieces of information.

```
sounds = ['boom','bang',
'kaboom','explode']

function boom() {
  z = random(sounds)
  sound(z)
  this.explode()
}
```

Notice that we are using a new **variable**, called **z**, to hold the randomly chosen sound. *We don't use quote marks to feed this variable into the **sound** command!*

CHALLENGING

Could it be possible to have different sounds play based on the direction the character was moving? To find out, we need to the **if/else** statement. This tests for a condition, does one thing if the condition is *true*, and another thing if the condition is *false*.

Remember the **x** and **y** built-in variables? If we click to the left of a **stamp**, **x** will be less than the **stamp's x** value. If we click below a **stamp**, **y** will be greater than the **stamp's y** value.

```
function touch() {
  if (x < a.x) {
    sound('walk')
    a.move(x,y,2000)
  }
  else {
    sound('walking')
    a.move(x,y,2000)
  }
}
function untouch() {
  a.move(a.x,a.y)
}
```

Try and also add the random sound effects to items, from the INTERMEDIATE column! You can also use the **silence** command if you need all sound effects to stop:

```
silence()
```

STAGE FIVE: COMPELLING SOUNDTRACK		
Add some background music to your app. You can click the  button for a list of songs that are included with Bitsbox.		
EASY	INTERMEDIATE	CHALLENGING

It's pretty easy to add a bit of music to an app; the **song** command will do it for you:
`song('name')`
 	
Songs can also be assigned a volume:
`song('name', number)`
 	
The number, here, is treated as a percentage; entering **20** will play the song at **20%** of maximum volume.
	Building off of the random selection of sound effects, why not populate an **array** with a bunch of song names, and then randomly choose one to play every time the app is started?	Building off of both the INTERMEDIATE and CHALLENGING portions of STAGE FOUR, populate two **arrays** with song names: one full of songs to play when the character isn't moving, and one to play when the character is moving.
		
		You'll need to use the **song** command in three places: once near the start of the app to give it an initial soundtrack, once within the **touch** function to define a song to play while the character is moving, and once within the **untouch** function to set the app back to a soundtrack for when the character isn't moving. And if you're feeling really keen, you can define soundtracks based on the *direction* the character is moving.

STAGE SIX: ITEM/CHARACTER INTERACTIONS

Make something happen when the character touches an item.

Every stamp has a (roughly) square **hitbox** that surrounds it; when these hitboxes meet, that's considered a touch. Note that it may not always look like stamps are actually touching!

EASY

If your character is still moving around the screen randomly (using the basic **move** command), it may occasionally touch one of the items you've placed in your app. The **hits** command will check if one **stamp** has touched another **stamp**, but you need to use the **if/else** statement to make this check happen. **if/else** tests for a condition, does one thing if the condition is *true*, and another thing if the condition is *false*.

Though, really, we only need to check the **if** part; we don't care if one **stamp** *isn't* touching the other:

```
if (a.hits(b)) {
  a.explode()
}
```

This is basically what happens to Mario every time he walks into a Goomba.

There's a problem, though: if you've only been using **move** or **rotate** without supplying any other information, Bitsbox won't actually detect it when **a** hits **b**. So, you need to go back and adjust how whatever is moving moves.

INTERMEDIATE

What if we added some text when a hit happens? It's one thing to make the item (or character) react to being hit, but sometimes we want to give extra visual feedback. To add text, we use the – what else? – **text** command.

```
if (a.hits(b)) {
  a.explode()
  text('OUCH!',a.x,a.y)
}
```

Bitsbox tracks the positions of things! For example, it will track where you touch or click on the tablet; the x and y values of where a touch event happens are stored in built-in **variables** called **x** and **y**.

But also, Bitsbox tracks the the x and y coordinates of each **stamp**; we use this above to make the text appear where the stamp that exploded used to be.

And since you are putting the **move** command inside a **function**, you can also put the **if** statement inside the same function (so that the program will check for a hit every time the **stamp** moves – this works *way* better!).

CHALLENGING

Suppose we populated an **array** with a list of the names of different kinds of effects – **explode**, **pop**, **splash**, and so forth. We can select one of those values out of the **array** and store it in a **variable**, but we can't apply the **variable** to the **stamp** like so:

```
effects = ['explode()',
'splash()','pop()']

effect = random(effects)
a.effect
```

THAT WON'T WORK!

We have to use **if/else** statements, instead. And, we have to make them **nested** (one inside another):

```
if (a.hits(b)) {
  if (effect == 'explode') {
    b.explode()
  }
  if (effect == 'splash') {
    b.splash()
  }
  if (effect == 'pop') {
    b.pop()
  }
  text('OUCH!',b.x,b.y)
}
```

EASY	INTERMEDIATE	CHALLENGING

If you supply the **move** command with numbers representing x and y coordinates, the stamp will instantly move to those coordinates when tapped:
`a.move(x,y)`
 	
You can also give a time value (in milliseconds) if you want the stamp to slide over to its new position:
`a.move(x,y,time)`
 	
And since you are putting the **move** command inside a **function**, you can also put the **if** statement inside the same function (so that the program will check for a hit every time the **stamp** moves – this works *way* better!).
		Now, do you think you can make the **text** say **random** things as well? You'll need to set up another **array** of phrases (nothing inappropriate!).

STAGE SEVEN: KEEPING THE GAME GOING

To this point, we've been pushing the  button to reset our app every time the character dies or the item(s) are gone. Let's figure out how to make the app reset itself!

This is what most games do, more or less: they run in a continuous loop, so that you don't have to constantly restart them every time you die.

INTERMEDIATE

When a **stamp** explodes, it doesn't actually get removed from the Bitsbox app! It's still there, but it is hidden automatically when the explosion animation plays (this is also true of splashes, pops, etc.).

Another fun fact: the **explode** command (as well as **pop**, **burn**, and the others that will make **stamps** disappear) can actually take the name of another function as an input. So, instead of this:

```
if (a.hits(b)) {
  a.explode()
}
```

We can do this:

```
if (a.hits(b)) {
  a.explode(tryagain)
}
```

And then we can build a new **function**:

```
function tryagain() {
  a.show()
  a.move()
}
```

This will cause the recently-exploded **stamp** to reappear and move to a random location on the screen.

CHALLENGING

Of course, it's one thing to make a single **stamp** disappear and reappear (e.g. to give a character both a death and a respawn). Do you think you can make every **stamp** in your project reappear after it disappears? And do you think you can do it in a way that one single **function** works for all **stamps** you are using?

Also, while we are on the subject, you've probably noticed that whatever stamp you are moving around the screen stays the same as it moves. That's weird, right? Fortunately, you can use the **flip** command in Bitsbox to make a stamp flip left or right.

```
a.flip()
```

Now, to insert this into your code in a way that won't look weird, you'll need to do a few things. If you aren't already checking position in the **touch function**, you'll want to head back to STAGE FOUR and set that up. But just doing this will produce odd results:

```
function touch() {
  if (x < a.x) {
    a.move(x,y,2000)
  }
  else {
    a.flip()
    a.move(x,y,2000)
  }
}
```

Testing this code out, you'll find that the **stamp** flips at times it shouldn't.

INTERMEDIATE	CHALLENGING

	To fix this, we need to use a **Boolean** (that is, a **variable** that is set to either **true** or **false**) to monitor which direction the **stamp** is facing.
	`facesright = false`
	`function touch() {`
	`  if (x < a.x) {`
	`  if (facesright == true) {`
	`  a.flip()`
	`  facesright = false`
	`  }`
	`  a.move(x,y,2000)`
	`  }`
	`  else {`
	`  if (facesright == false) {`
	`  a.flip()`
	`  facesright = true`
	`  }`
	`  a.move(x,y,2000)`
	`  }`
	`}`
	
	In case you're wondering why we use one equals sign (=) in some places, and a double equals sign (==) in others, the difference is this: a single equals sign is for *assigning* a value to a **variable**, and a double equals sign is for *comparing* the value of a **variable** to another variable or a set value.

STAGE EIGHT: SCORING

What's a game without some way of tracking how well the player is doing at it?

We've been using **text** in a few places already, but now we are going to make it work for us.

CHALLENGING

In Bitsbox, **text** can – like a **stamp** – be assigned a position on the screen by supplying it with x and y coordinates. It can also be assigned to a **variable**. Remember when we did this?

```
if (a.hits(b)) {
  a.explode()
  text('OUCH!',a.x,a.y)
}
```

If you've implemented a basic game loop, you've probably noticed that there's a lot of text getting left all over the screen. So, as fun as it has been messing up the screen, we're going to try do something a little different.

```
t = text('HITS: ',x,y,color)
```

Notice that in addition to coordinates, we can also tell the **text** to be a different color! This will be useful when the text is over backgrounds that might otherwise make it hard to see.

Now, here we are tracking a count of hits. We need another **variable** to count those:

```
hitcount = 0
```

And we need to include the count in the text we are displaying. Bitsbox gives us the **+** operator to make this happen:

```
t = text('HITS: ' + hitcount,x,y,color)
```

However, just putting this line of code into the program won't actually make the hit counter increment, for two reasons. One, we haven't actually told the code where to count a hit, and two, we aren't telling the **text** to update.

It's pretty easy to figure out where to count hits:

```
if (a.hits(b)) {
  a.explode()
  hitcount = hitcount + 1
}
```

Notice what we do there: we take hitcount, and set it equal to *itself plus one*.

CHALLENGING

Now, how do we make the text show that? There's one more command we need, the **change** command.

```
if (a.hits(b)) {
  a.explode()
  hitcount = hitcount + 1
  t.change('HITS: ' + hitcount)
}
```

Notice that when we use the **change** command with the **text**, we only give it the updated **text** to display, and we *do not* give it coordinates or a color. We can give it these values too, but if we don't, then the **text** will keep the coordinates and color we assigned it previously.

STAGE NINE: EXPANDING THE WORLD

It's a bit boring playing a game with a small world. Find a way to expand yours!

CHALLENGING

THIS IS A SIGNIFICANT CHALLENGE; ONLY ATTEMPT IF YOU HAVE AT LEAST HALF AN HOUR

There will be no sample code in this section. Instead, what are a couple of ways that we can add some change to the world of the game we've been building all along?

In general, there are two kinds of transitions that will be used in games: *scrolling* and *warps*. Scrolling is a *continuous movement through a space*, while warps are *instantaneous jumps between different spaces*. If you think about a Mario game, you'll find examples of both: as you're playing through World 1-1, the level will scroll past as you move through it. But when you're done, you'll be warped to World 1-2.

Suppose you have a character **stamp** in your Bitsbox app, and two item **stamps**. One way of implementing a warp – to make it seem like the character has moved to a different level – would be to change the background when the character does something, and maybe also change one or both of the items as well.

Conversely, for a good example of a scrolling app, ask Akela for a copy of *Super Plumber Jumper*; that app shows how to create a series of platforms in an app, which a character has to jump between. As the character jumps "out of" the top of the tablet, the view scrolls up to show new platforms.

SAMPLE CODE: EASY

This is an example program using code that is discussed in the EASY column for STAGES ONE through SIX.

There is no EASY code for STAGES SEVEN through NINE.

```
// STAGE FIVE
song('village', 15)

// STAGE ONE
a = stamp('wolf', 516, 668)
b = stamp('steak3', 166, 862, 150)
c = stamp('lettuce2', 296, 352, 150)
fill('campground')

// STAGE TWO
// b.tap = pop
// c.tap = splash

// STAGE THREE
// function zoom() {
//  a.move()
// }
a.tap = zoom

// STAGE FOUR
function sploosh() {
  sound('splash')
  this.splash()
}
function kapow() {
  sound('pop')
  this.pop()
}
b.tap = kapow
c.tap = sploosh

// STAGE SIX
function zoom() {
  a.move(x, y, 500)
  if (a.hits(b)) {
   sound('chomp')
   b.pop()
  }
  if (a.hits(c)){
   sound('howl')
   a.explode()
  }
}
```

SAMPLE CODE: INTERMEDIATE

This is an example program using code that is discussed in the INTERMEDIATE column for STAGES ONE through SEVEN.

There is no INTERMEDIATE code for STAGES EIGHT and NINE.

```
// STAGE FIVE
soundtrack = ['village','grass','farm']
tune = random(soundtrack)
song(tune, 20)

// STAGE ONE
a = stamp('wolf', 516, 668)
b = stamp('steak3', 166, 862, 150)
c = stamp('lettuce2', 296, 352, 150)
fill('campground')

// STAGE TWO
// function sploosh() {
//   this.splash()
// }
// function kapow() {
//   this.pop()
// }
b.tap = kapow
c.tap = sploosh

// STAGE THREE
// function zoom() {
//  a.move(x, y, 500)
// }
a.tap = zoom

// STAGE FOUR
popsounds = ['pop','pop2','balloon']
splashsounds = ['splash','drip','splotch']
// function sploosh() {
//   z = random(splashsounds)
//   sound(z)
//   this.splash()
// }
// function kapow() {
//   z = random(popsounds)
//   sound(z)
//   this.pop()
// }

// STAGE SIX
// function zoom() {
//  a.move(x, y, 500)
//  if (a.hits(b)) {
//    sound('chomp')
```

```
//    b.pop()
//    text('YUMMY!', b.x, b.y)
//  }
//  if (a.hits(c)){
//    sound('howl')
//    a.explode()
//    text('UGH!', a.x, a.y)
//  }
// }

// STAGE SEVEN
function zoom() {
  a.move(x, y, 500)
  if (a.hits(b)) {
    sound('chomp')
    b.pop()
    text('YUMMY!', b.x, b.y)
  }
  if (a.hits(c)){
    sound('howl')
    a.explode(tryagain)
    text('UGH!', a.x, a.y)
  }
}
function sploosh() {
  z = random(splashsounds)
  sound(z)
  this.splash()
}
function kapow() {
  z = random()
  sound(z)
  this.pop()
}
function tryagain() {
  a.show()
  a.move()
}
```

SAMPLE CODE: CHALLENGING

This is an example program using code that is discussed in the CHALLENGING column for STAGES ONE through EIGHT.

STAGE NINE is left as a challenge for you to solve.

```
// STAGE FIVE
soundtrack = ['village','grass','farm']
tune = random(soundtrack)
song(tune, 20)

// STAGE ONE
a = stamp('wolf', 516, 668)
b = stamp('steak3', 166, 862, 150)
c = stamp('lettuce2', 296, 352, 150)
fill('campground')

// STAGE TWO
// function sploosh() {
//   this.splash()
// }
// function kapow() {
//   this.pop()
// }
b.tap = kapow
c.tap = sploosh

// STAGE THREE
// function tap() {
//   a.move(x, y, 500)
// }
// function touch() {
//   a.move(x, y, 1000)
// }
// function untouch() {
//   a.move(a.x, a.y)
// }

// STAGE FOUR
popsounds = ['pop','pop2','balloon']
splashsounds = ['splash','drip','splotch']
// function touch() {
//   if (x < a.x) {
//     sound('walk')
//     a.move(x, y, 5000)
//   }
//   else {
//     sound('walking')
//     a.move(x, y, 5000)
//   }
// }
// function untouch() {
```

```
//   a.move(a.x, a.y)
// }
function sploosh() {
  z = random(splashsounds)
  sound(z)
  this.splash(tryagain)
}
function kapow() {
  z = random(popsounds)
  sound(z)
  this.pop(tryagain)
}

// STAGE FIVE
westsongs = ['western','westward']
eastsongs = ['east','dawn']
// function touch() {
//   if (x < a.x) {
//     silence()
//     sound('walk')
//     song(random(eastsongs), 20)
//     a.move(x, y, 5000)
//   }
//   else {
//     silence()
//     sound('walking')
//     song(random(westsongs), 20)
//     a.move(x, y, 5000)
//   }
// }
function untouch() {
  silence()
  song(tune, 20)
  a.move(a.x, a.y)
}

// STAGE SIX
effects = ['explode','burn','pow']
phrases = ['UGH!','YUCK!','OH NO!']
effect = random(effects)
lastwords = random(phrases)
// function touch() {
//   if (x < a.x) {
//     silence()
//     sound('walk')
//     song(random(eastsongs), 20)
//     a.move(x, y, 2500)
//     if (a.hits(b)) {
//       sound('chomp')
//       b.pop()
//       text('YUMMY!', b.x, b.y)
```

```
//     }
//     if (a.hits(c)){
//       sound('howl')
//       if (effect == 'explode') {
//         a.explode()
//       }
//       if (effect == 'burn') {
//         a.burn()
//       }
//       if (effect == 'pow') {
//         a.pow()
//       }
//       text(lastwords, a.x, a.y)
//     }
//   }
//   else {
//     silence()
//     sound('walking')
//     song(random(westsongs), 20)
//     a.move(x, y, 2500)
//   }
// }

// STAGE SEVEN
facesright = false
function tryagain() {
  this.show()
  this.move()
}
// function touch() {
//   if (x < a.x) {
//     silence()
//     sound('walk')
//     song(random(westsongs), 20)
//     if (facesright == true) {
//       a.flip()
//       facesright = false
//     }
//     a.move(x, y, 2500)
//     if (a.hits(b)) {
//       sound('chomp')
//       b.pop(tryagain)
//       text('YUMMY!', b.x, b.y)
//     }
//     if (a.hits(c)){
//       sound('howl')
//       if (effect == 'explode') {
//         a.explode(tryagain)
//       }
//       if (effect == 'burn') {
//         a.burn(tryagain)
```

```
//       }
//       if (effect == 'pow') {
//         a.pow(tryagain)
//       }
//       text(lastwords, a.x, a.y)
//     }
//   }
//   else {
//     silence()
//     sound('walking')
//     song(random(eastsongs), 20)
//     if (facesright == false) {
//       a.flip()
//       facesright = true
//     }
//     a.move(x, y, 2500)
//   }
// }

// STAGE EIGHT
score = 0
deaths = 0
scoretext = text('EATEN: ' + score, 24, 36, 'white')
deathtext = text('DIED: ' + deaths, 539, 36, 'white')
function touch() {
  if (x < a.x) {
    silence()
    sound('walk')
    song(random(westsongs), 20)
    if (facesright == true) {
      a.flip()
      facesright = false
    }
    a.move(x, y, 2500)
    if (a.hits(b)) {
      sound('chomp')
      b.pop(tryagain)
      score = score + 1
      scoretext.change('EATEN: ' + score)
    }
    if (a.hits(c)){
      sound('howl')
      if (effect == 'explode') {
        a.explode(tryagain)
        deaths = deaths + 1
        deathtext.change('DIED: ' + deaths)
      }
      if (effect == 'burn') {
        a.burn(tryagain)
        deaths = deaths + 1
        deathtext.change('DIED: ' + deaths)
```

```
      }
      if (effect == 'pow') {
        a.pow(tryagain)
        deaths = deaths + 1
        deathtext.change('DIED: ' + deaths)
      }
    }
  }
  else {
    silence()
    sound('walking')
    song(random(eastsongs), 20)
    if (facesright == false) {
      a.flip()
      facesright = true
    }
    a.move(x, y, 2500)
  }
}
```