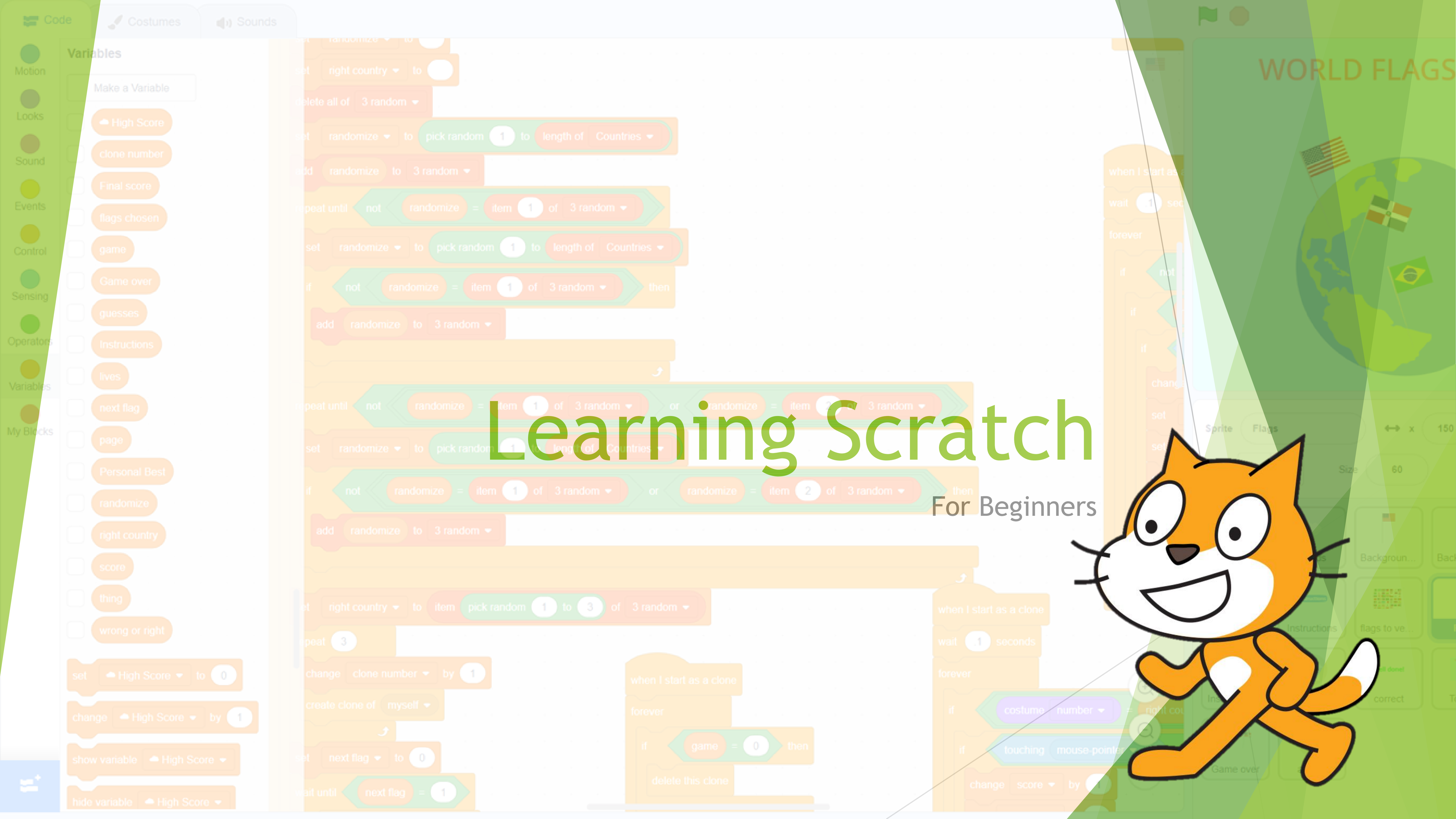


Learning Scratch

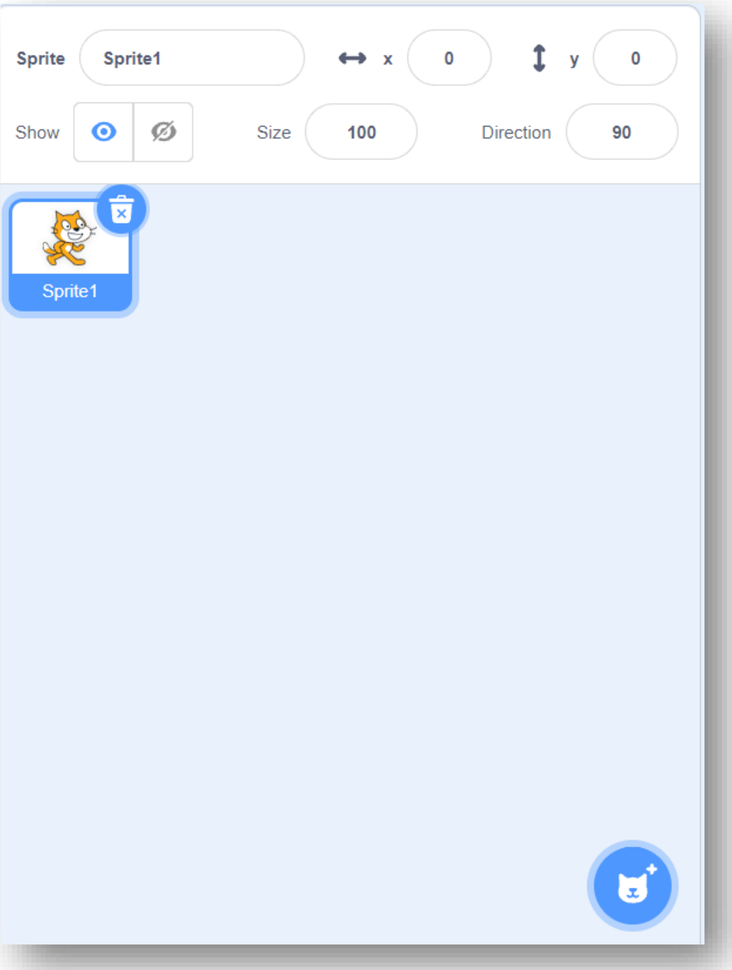
For Beginners

What is Scratch?

- Scratch is basic coding language used to make games, animations, and more!
- Scratch does not require you to understand confusing syntax, (how to properly write code) but instead allows you to drag blocks together. This makes for a fun, easy to understand experience.
- For today’s lesson we will be using the scratch 3.0 offline editor, which we have downloaded on your computers

Breaking down the scratch 3.0 interface (1)

Once we open scratch, we see this on the bottom right corner of our screen:



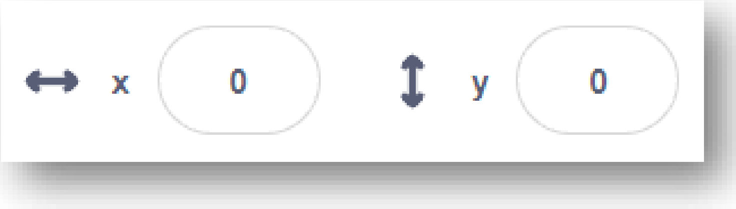
Sprite Sprite1 ↔ x 0 ↕ y 0

Show Size 100 Direction 90

Here, we see a *Sprite*. Simply put, a scratch sprite is a character that is affected by the code we make. Right now, we only have one sprite, the default cat sprite which scratch gives us. Above this sprite, are its attributes (its name, position on the screen, whether its being shown or not, its size, and the direction its pointing in)

Breaking down the scratch 3.0 interface (1.1)

Our cat’s exact position on the screen is shown with 2 numbers, both 0 in in this case. But what do they mean?

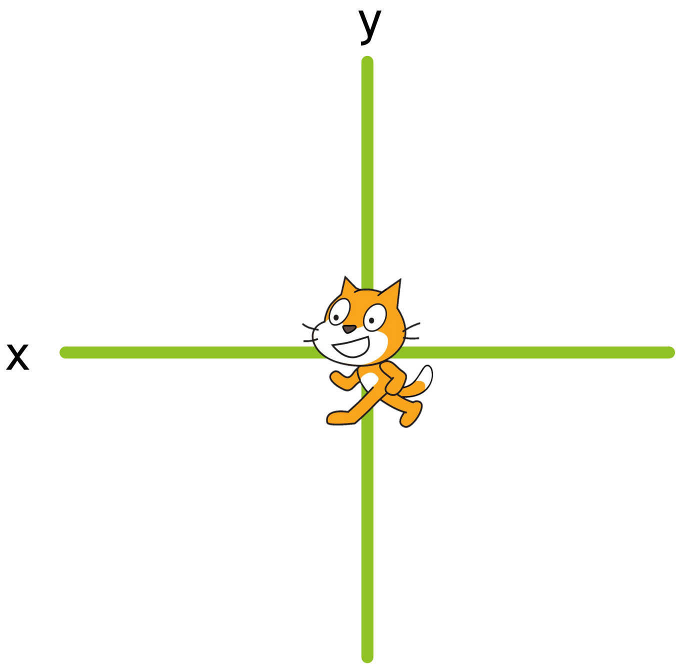


↔ x 0 ↕ y 0

The first number known as the *x value* shows where the sprite is horizontally, meaning from side to side.

The second number, known as the *y value* shows where the sprite is vertically, meaning up and down.

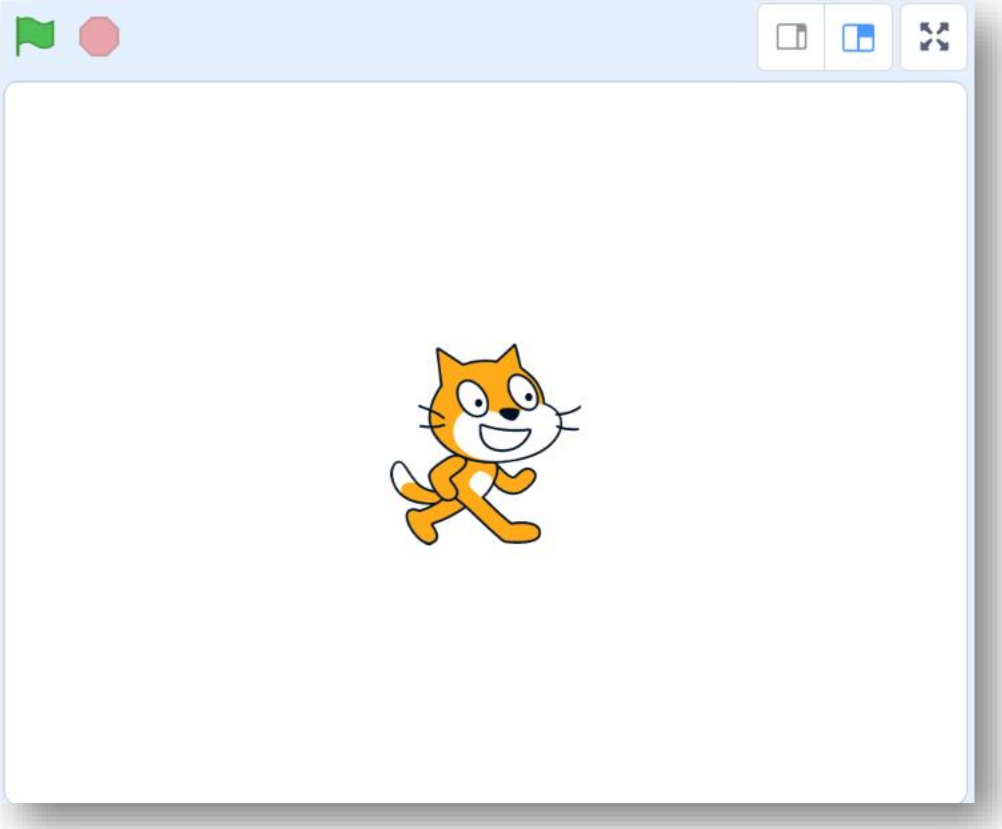
Right now, our sprite sits at x=0, y=0, meaning it is at the center of the screen



If this is not making sense yet, don’t worry! It will become clearer as we start working on our first game.

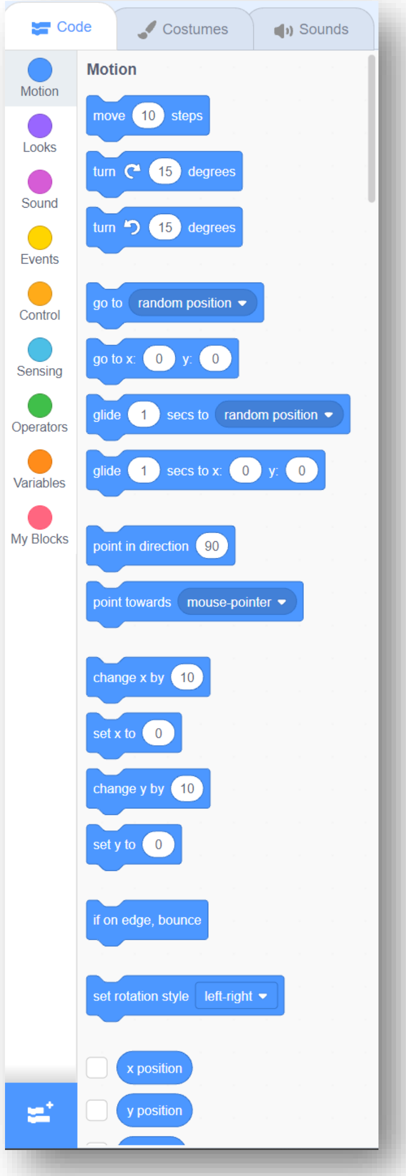
Breaking down the scratch 3.0 interface (2)

Just above our list of sprites and their attributes, we see the screen where our sprites will move and react as instructed by our code.



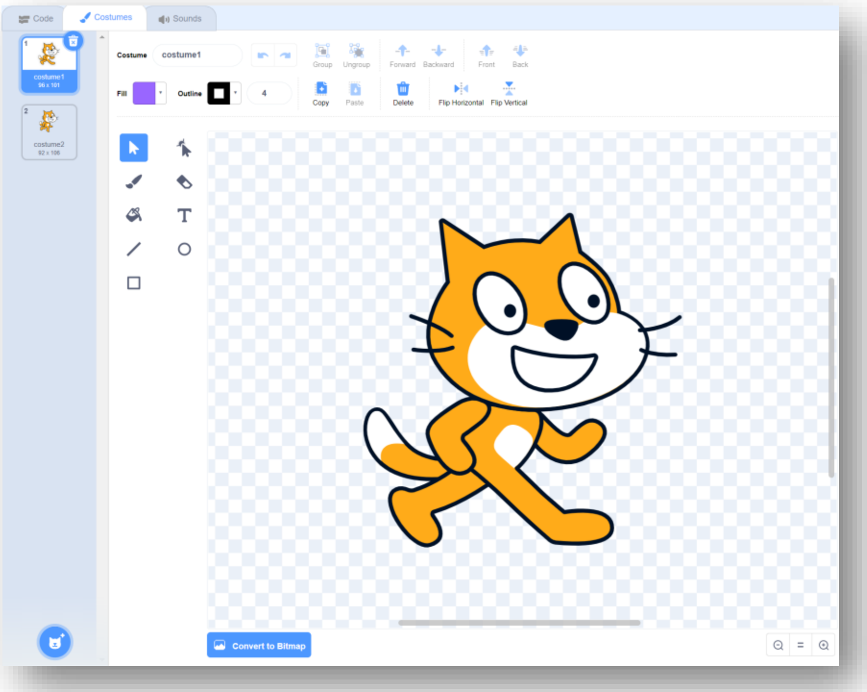
On the upper left corner, we see a green flag. When this flag is clicked, it will start your project, and run the code. Next to it, is a red stop button, that will stop your project when it is clicked. The buttons on the upper right only change how we see this screen, so they are not important.

Breaking down the scratch 3.0 interface (3)



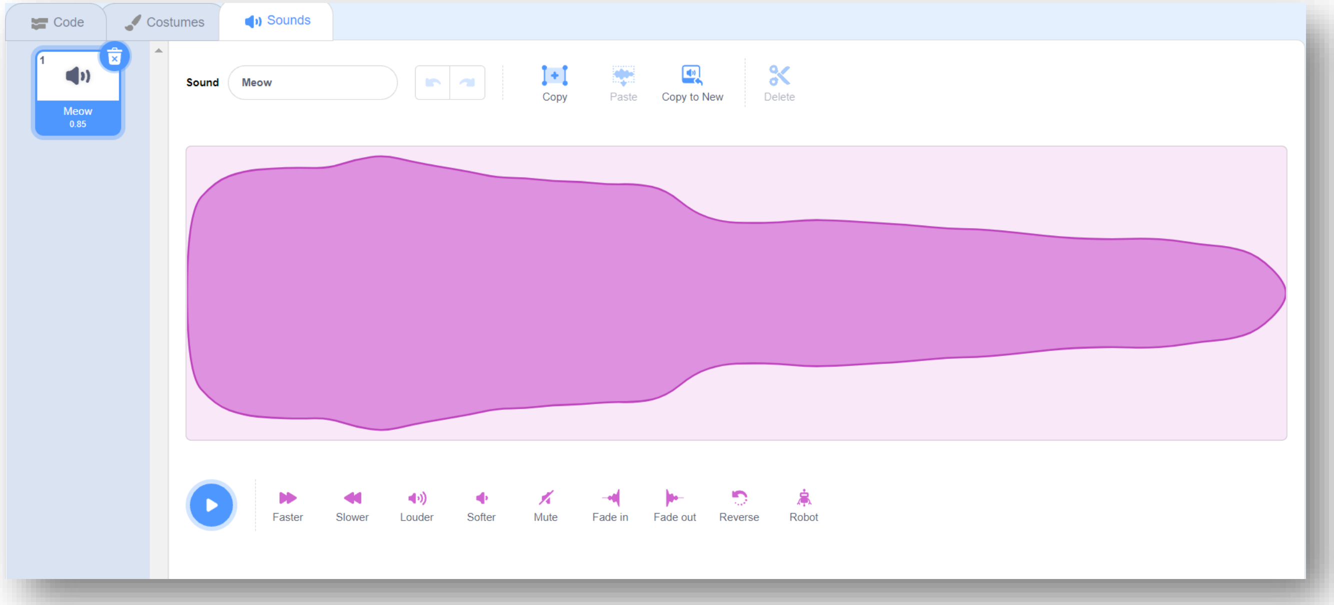
On the far left side, we find all of the blocks we will use to code for our sprite. The different sections of blocks (motion, looks, sounds, etc.) all effect the sprite in different ways. We will learn how some of these blocks work when we start working on our first game.

Breaking down the scratch 3.0 interface (4)



If we switch over from the “code” tab to the “costumes” tab, we are greeted with our cat sprite again, in an art editor. Costumes are like sprites, but are contained within them. One sprite may have many different costumes. Simply put, a costume is an appearance for a sprite.

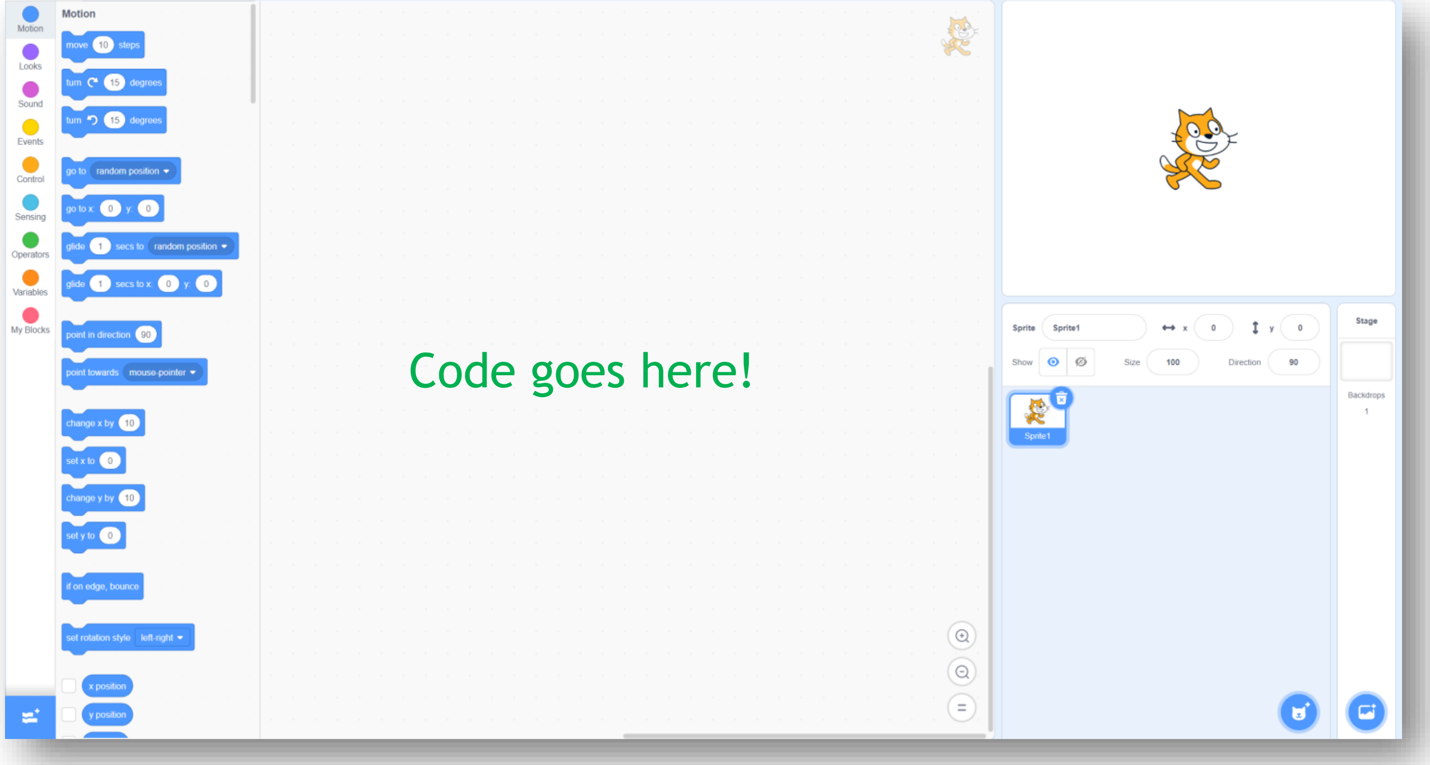
Breaking down the scratch 3.0 interface (5)



Finally, if we switch to the "sounds" tab, we see a pink shape which is a visual representation of audio. While it is possible to add custom sounds to your projects, we won't be doing that today.

Breaking down the scratch 3.0 interface (6)

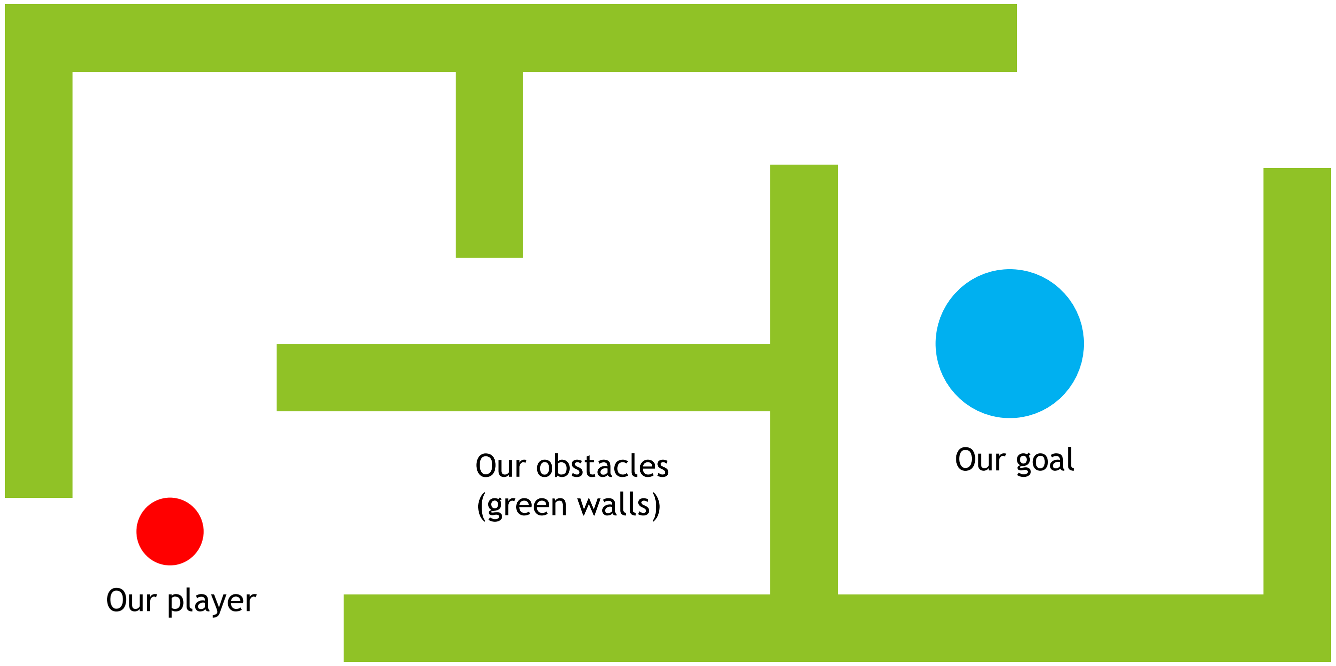
And to conclude this overview of the scratch 3.0 interface, we see a large empty space in the middle of the screen. This is where the code for a sprite goes.



Code goes here!

Maze Game

For our first project in scratch, we will be making a maze game. In the game we will move an object around on screen using the arrow keys to avoid walls and other obstacles, eventually reaching the end of the level, and moving on to the next level.



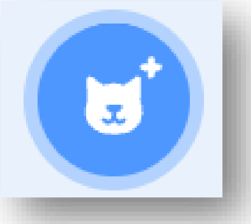
Our obstacles (green walls)

Our goal

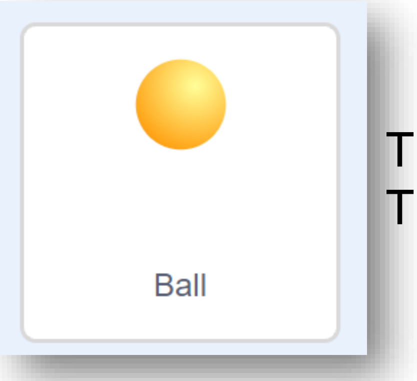
Our player

Selecting our player sprite

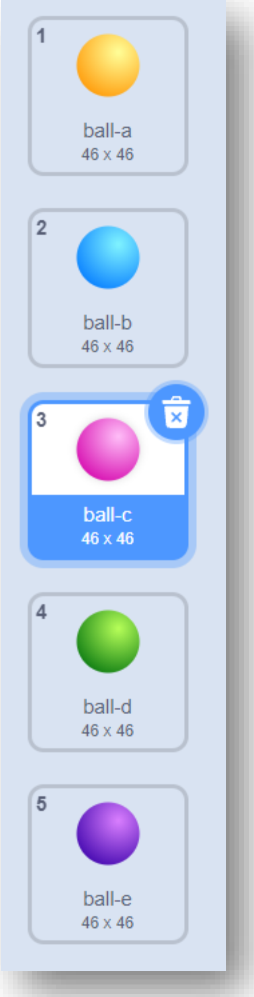
Let’s begin by finding a sprite to use for our player. While we could make the sprite using scratch’s art editor, we will instead choose it from scratch’s large sprite library.



Click this button in the bottom right corner of the screen.



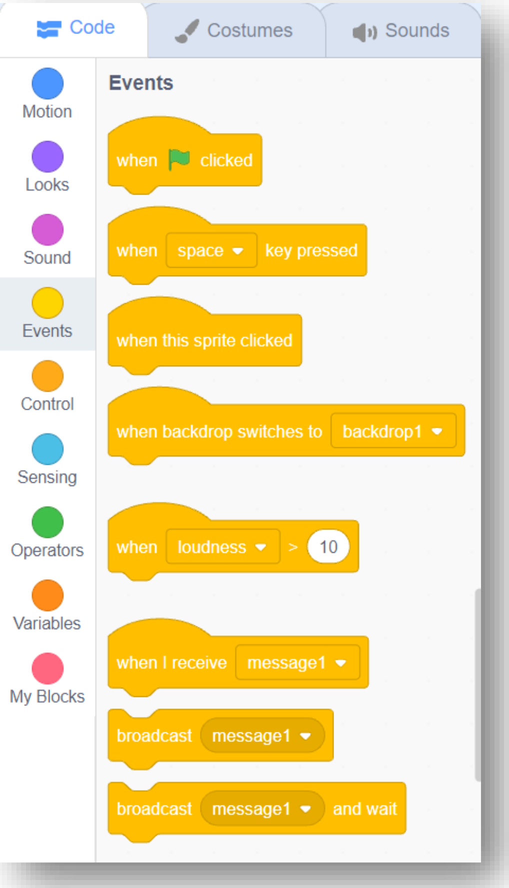
Then, click on the “ball” sprite. This will add it to our project.



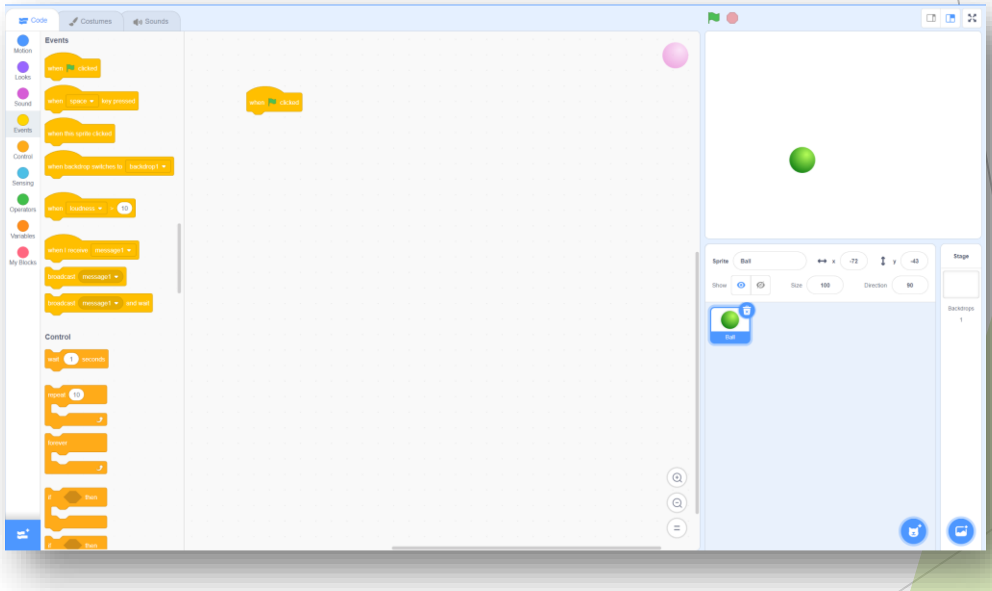
When we open the “costumes” tab for the ball sprite, we’ll notice there a few different costumes for it. Delete 4 of them by clicking the small icon in their top right corner and keep the one of your choice.

Starting to code

Let’s begin by making our sprite have a starting position when we start the project (remember the project is started by clicking the green flag). In order to do this, we will use the “When flag clicked” block in the “events” category,

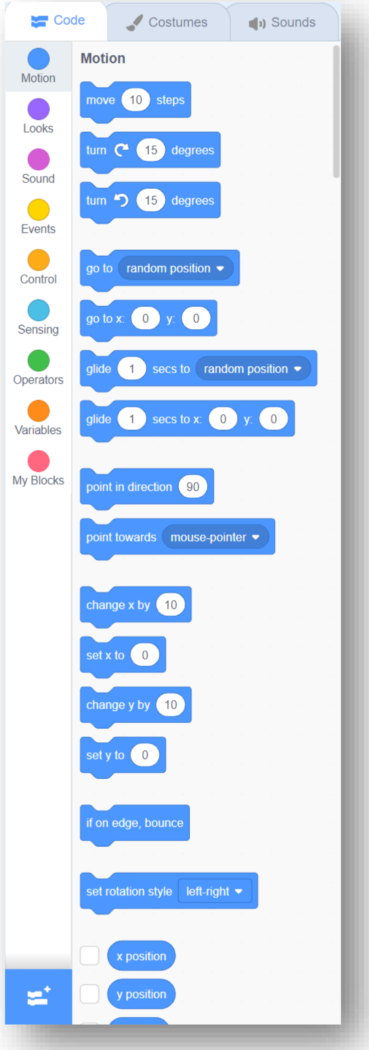


Drag it into the blank canvas

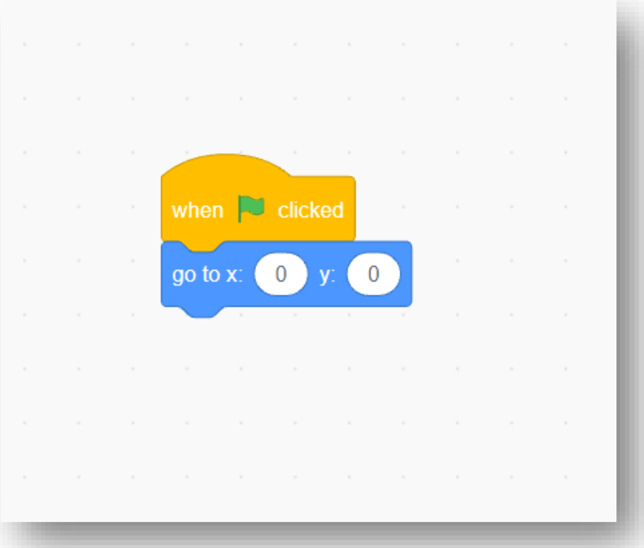


Position

Now we want to move the ball, the player sprite, to a starting location on the screen. To do this, we will use a block from the "motion" category.



Drag out the "go to x: , y: block

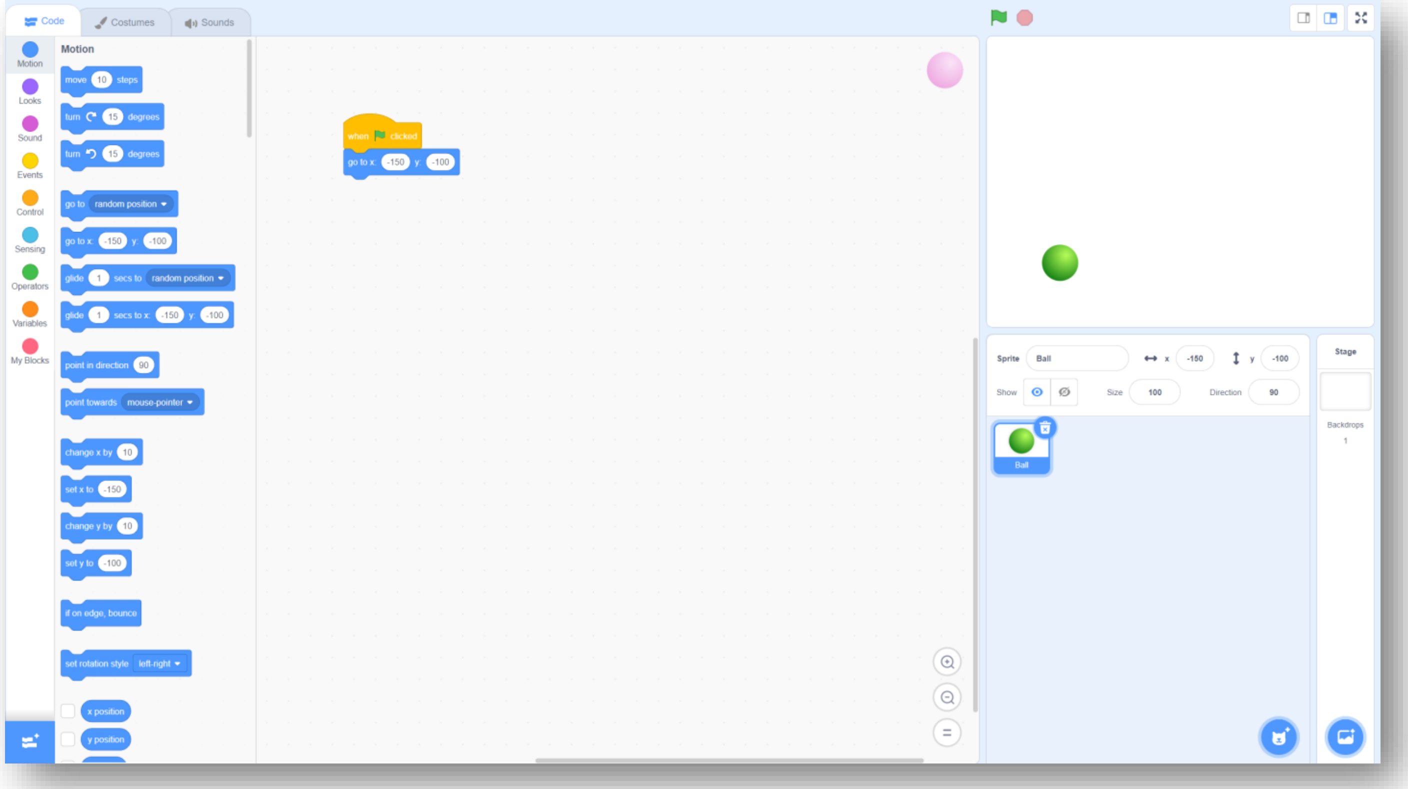


```
when 🏁 clicked
go to x: 0 y: 0
```

Attach it to the "when flag clicked" block by placing it slightly under.

Position (2)

While it is not especially important to understand how the x position and y position work, have fun putting in random values for x and y, then clicking the green flag and seeing where the ball moves!



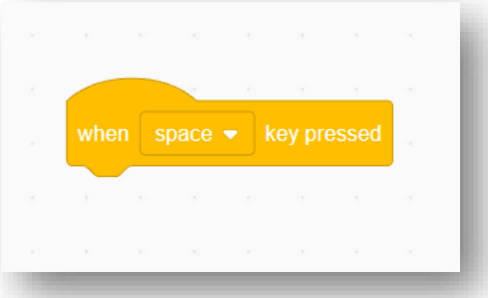
```
when [flag] clicked
go to x: -150 y: -100
```

Sprite: Ball x: -150 y: -100
Show Size: 100 Direction: 90

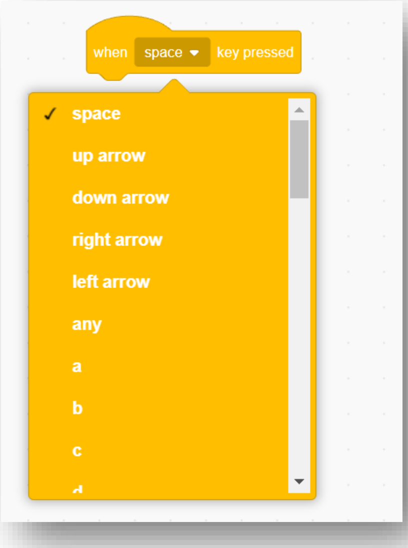
I find that setting the x value to -150, and y value to -100 places the ball nicely in the bottom left corner of our screen. A perfect starting placement for our game.

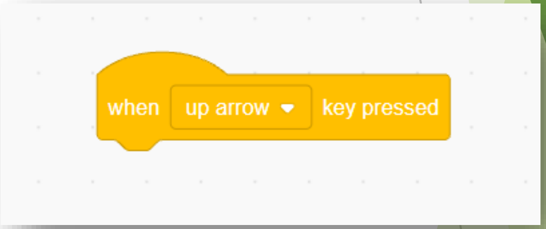
Controlling the sprite

Now that we have a starting position for our sprite, we can work on getting it to move in response to the arrow keys. Luckily, scratch has a block just for responding to keys. In the “events” category, just below the flag block, is what we’re looking for. Drag it onto the canvas.



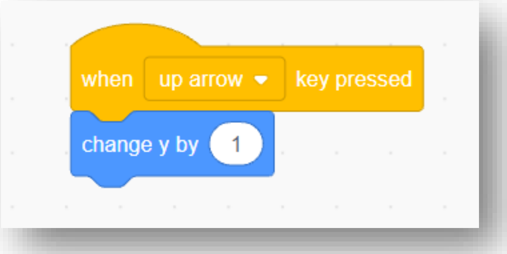
unroll the dropdown menu, and select the up arrow key!





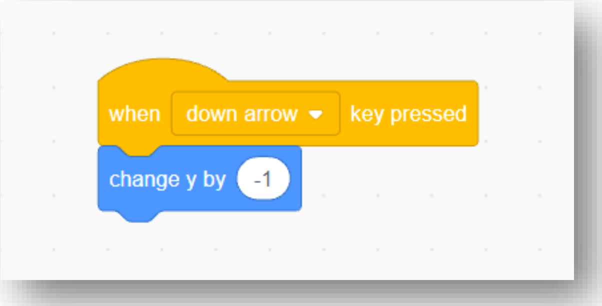
Controlling the sprite (2)

Now that we have a block ready to respond to the up arrow key, we need to put a block below it which will tell the sprite what to do. In this case, we want to move up, so we will use the “change y by” block in the motion category.



```
when up arrow ▾ key pressed
change y by 1
```

We’ve placed the block underneath our “when up arrow key pressed block. So when the up arrow key is pressed, the position of our sprite will go up the screen 1 pixel at a time.

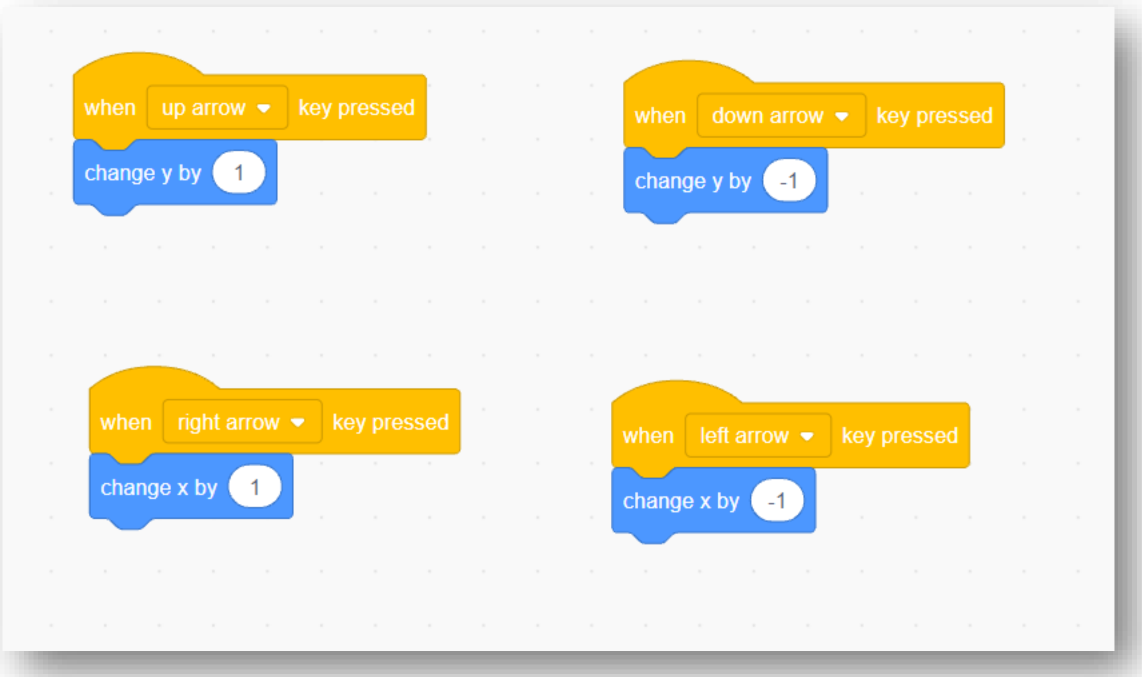


```
when down arrow ▾ key pressed
change y by -1
```

Let’s drag out the same blocks again, but this time causing the down arrow to make the player move down.

Controlling the sprite (3)

If we continue this process, considering that up = +y, down = -y, right = +x, and left = -x, We will end up with the following four scripts.

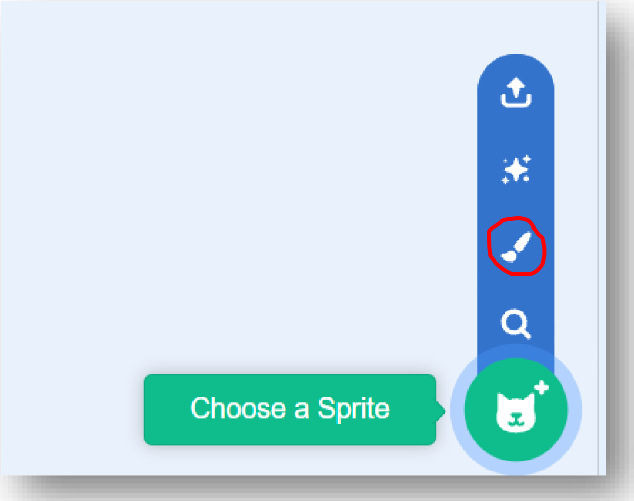


```
when up arrow ▾ key pressed
change y by 1

when down arrow ▾ key pressed
change y by -1

when right arrow ▾ key pressed
change x by 1

when left arrow ▾ key pressed
change x by -1
```

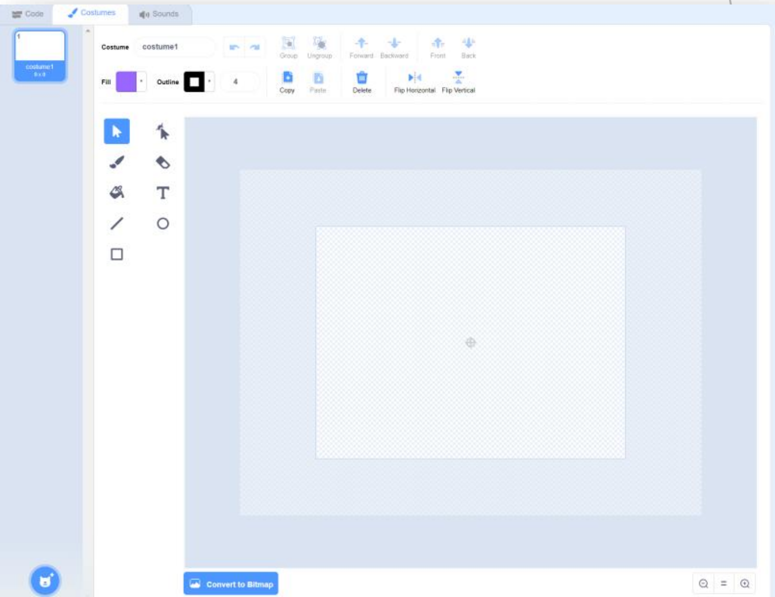
Try using the arrow keys to move around the sprite!

Making the maze

Now lets take a break from the coding, and draw our maze. The maze will be made of rectangles and squares which our player will have to avoid.



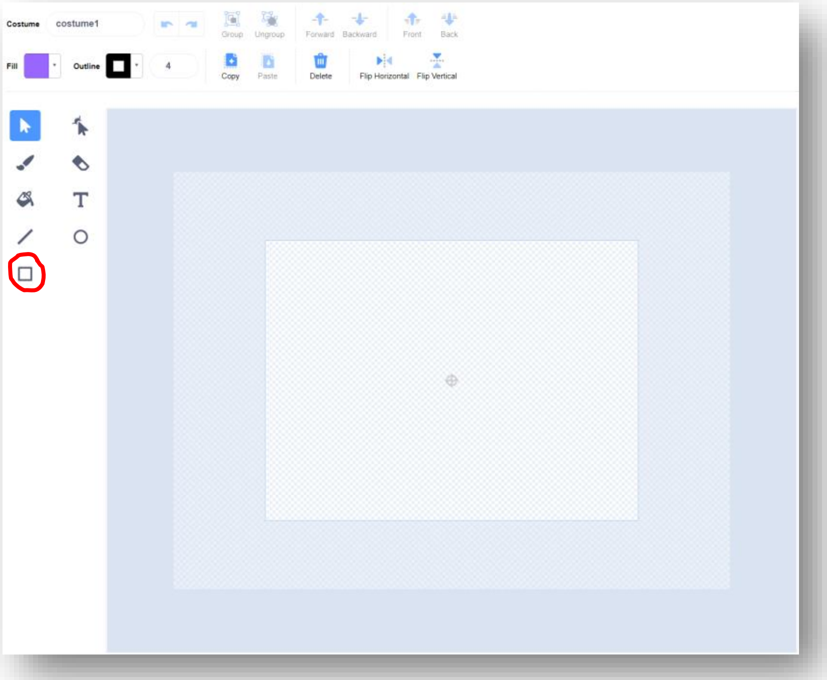
Let’s create a new sprite, and instead of choosing one from scratch’s library, we’ll be making our own by selecting the paint-brush icon.



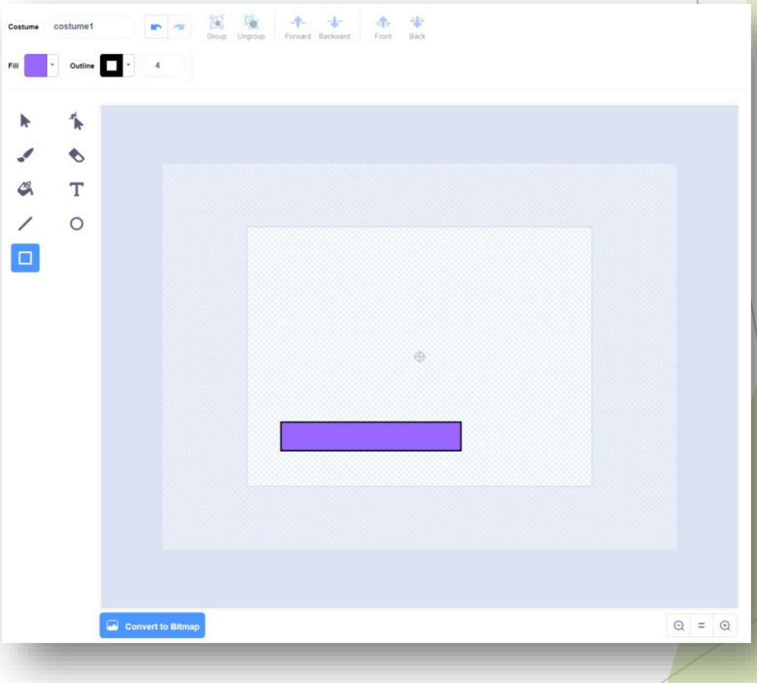
We now have a nice blank canvas to work on.

Making the maze (2)

For this maze, we will be using the “square” tool in scratch’s art editor.

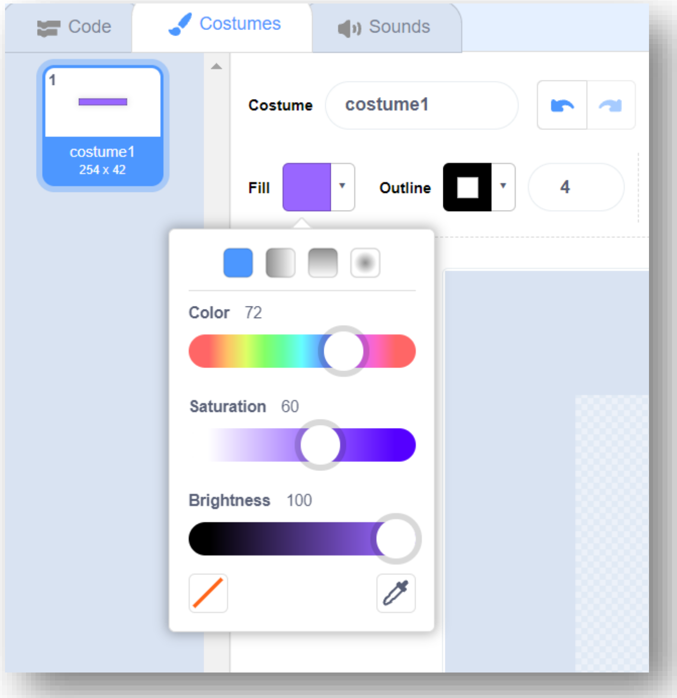


Click + Drag



Making the maze (3)

But wait! Do you really want your maze’s walls to be purple and black? Probably not.



Costume costume1

Fill

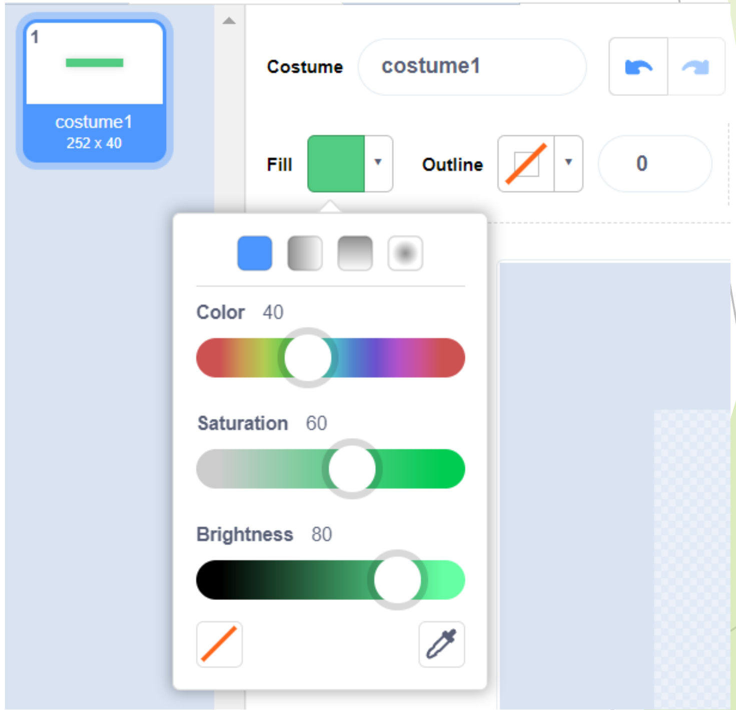
Outline 4

Color 72

Saturation 60

Brightness 100

Unroll the dropdown menus, and experiment with the dials until you get a color you like! (do this while your rectangle you just drew is selected)



Costume costume1

Fill

Outline 0

Color 40

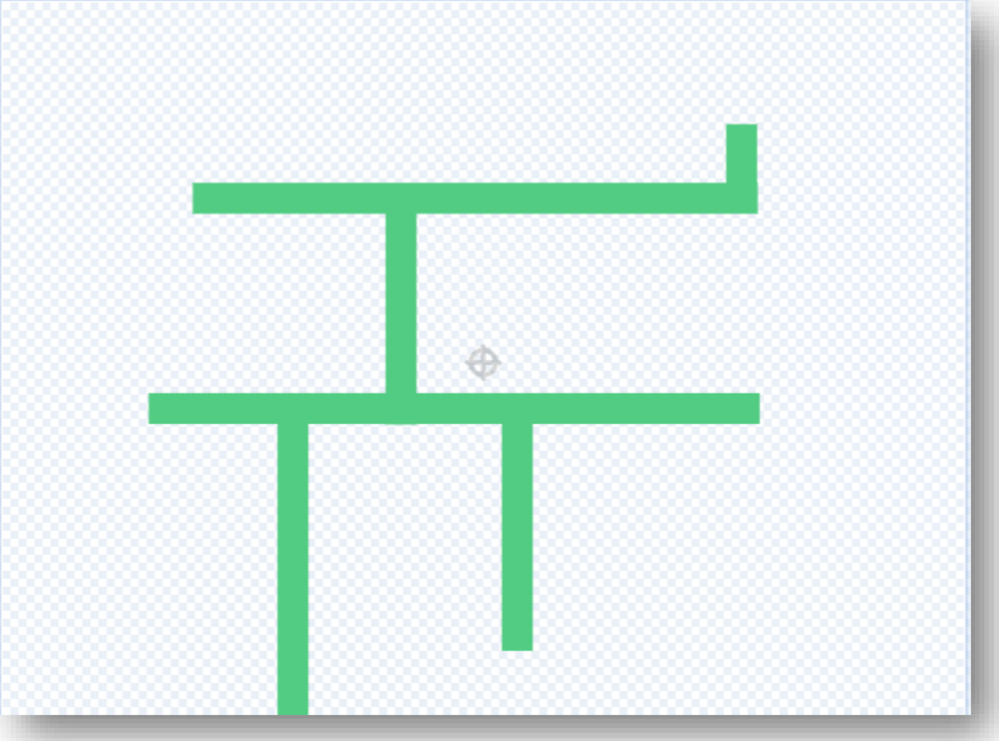
Saturation 60

Brightness 80

I like this green color, with no outline, but you can choose whatever you want!

Making the maze (4)

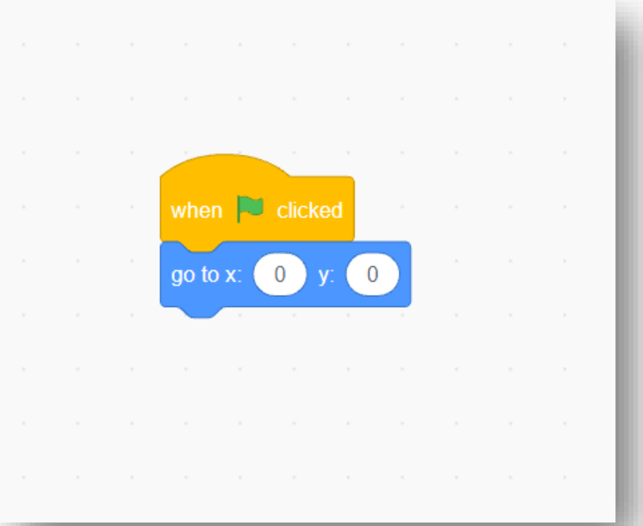
Once we’ve selected a color to work with, let’s start creating the maze! Make sure to avoid putting walls near the bottom left corner, though, because our player will be starting there. We don’t want to run into a wall as soon as we begin!



Here is my basic maze design, made of rectangle walls.

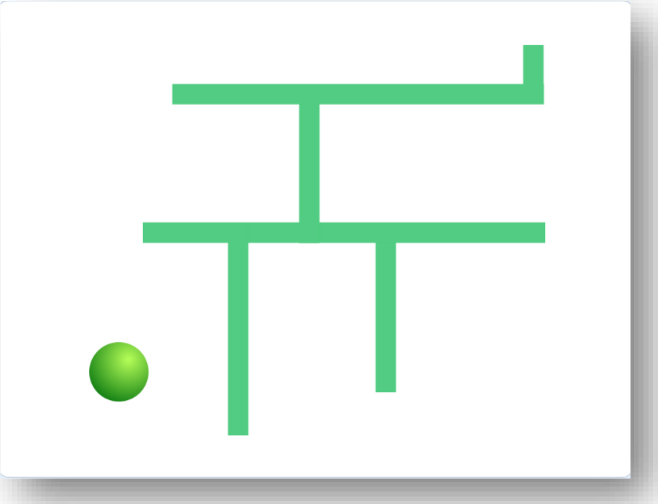
Making the maze (5)

Lets make sure our maze is in the center of the screen by using the following script which we used before.



```
when 🏁 clicked
go to x: 0 y: 0
```

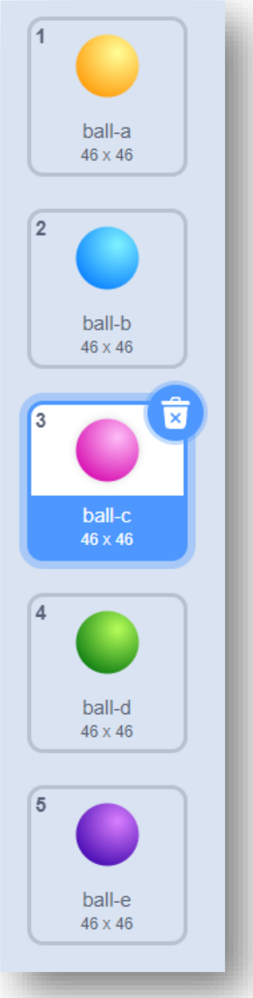
This script, when put in our maze sprite, will make it go to the center of the screen.



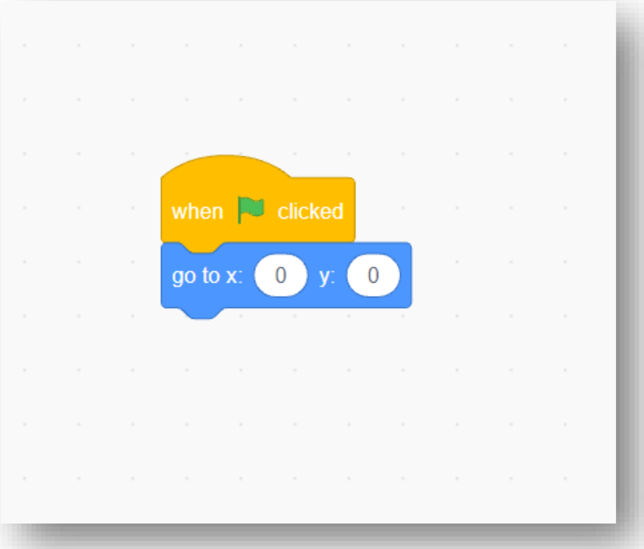
Now lets see what our screen looks like for someone playing the game!

The maze goal

Now that we have a player sprite and a maze sprite, the only other sprite we need is the goal for the player to get to. Once we make that, we can work on coding the walls so that they send the player back to the start of the maze it touched.



Lets grab another ball sprite from the scratch sprite library, and once again delete all its costumes but one (preferably don't use the same costume as the player sprite)

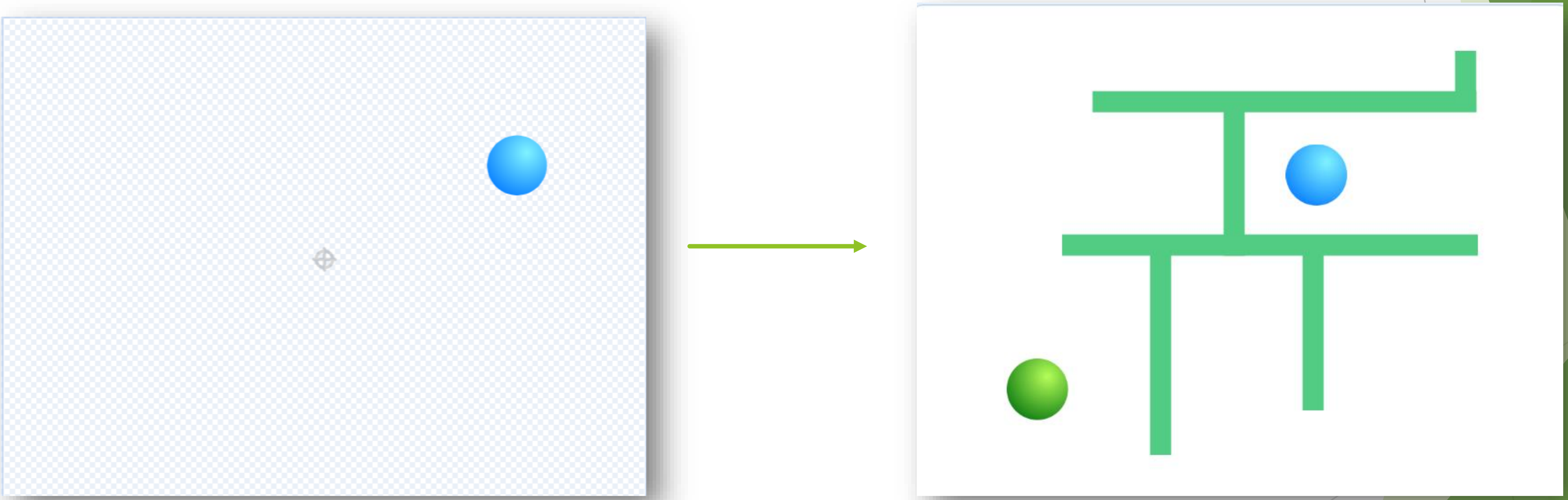


```
when 🏁 clicked
go to x: 0 y: 0
```

Let's make sure our sprite is at the center of screen using this script again.

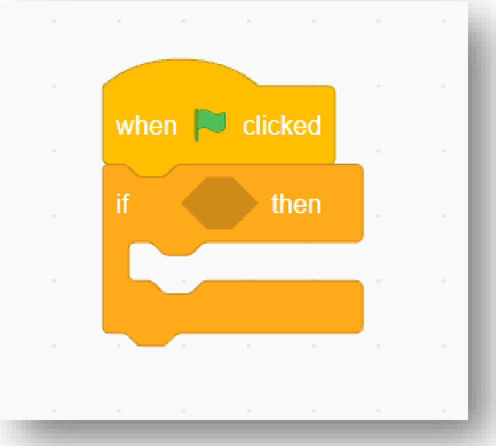
The maze goal (2)

Now all we need to do is move around our goal sprite inside the art editor, until we find a good place for it to be within the maze. You will see the 1 to 1 correspondence between the editor and the screen our player will see, when moving it around.

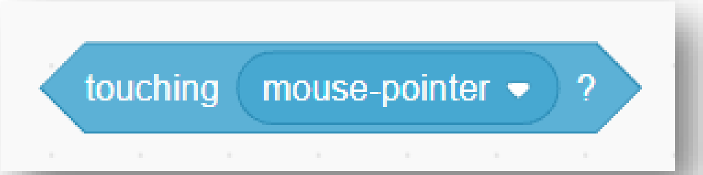


Collisions

Now we will make the player return to the start of the maze if it touches a wall. Let's drag out our "when flag clicked block" and put a "if then" block under it. Remember we're coding in our player sprite.



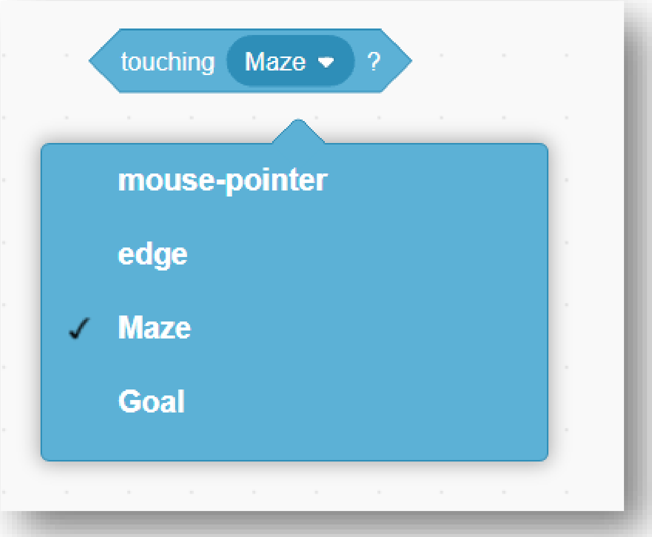
This new block is found in the "control" category, and allows us to run certain code if a condition is met.



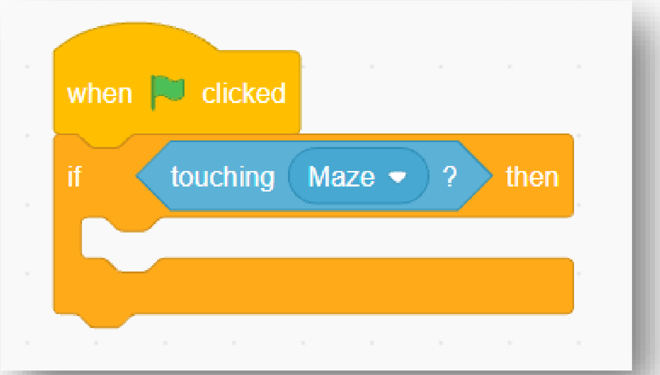
Now lets drag out a "touching" block from the top of the "sensing" category. We will combine these scripts later.

Collisions (touching block and sprite names)

You may notice that the “touching” block is currently on “mouse-pointer” which means it is testing to see if the sprite is touching your mouse. This is not what we want, so let’s unroll the dropdown menu and change it.



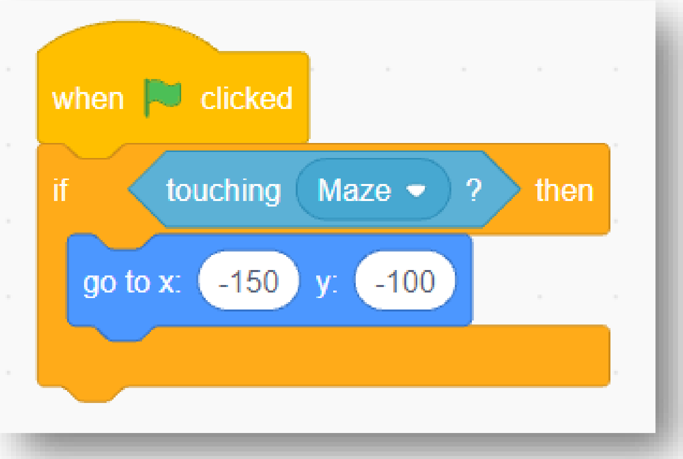
If you haven't already, make sure to change your sprite names to accurately reflect the sprite. I have now selected “maze”.



Now we put the “touching” block into the “if then” block.

Collisions (a mistake?)

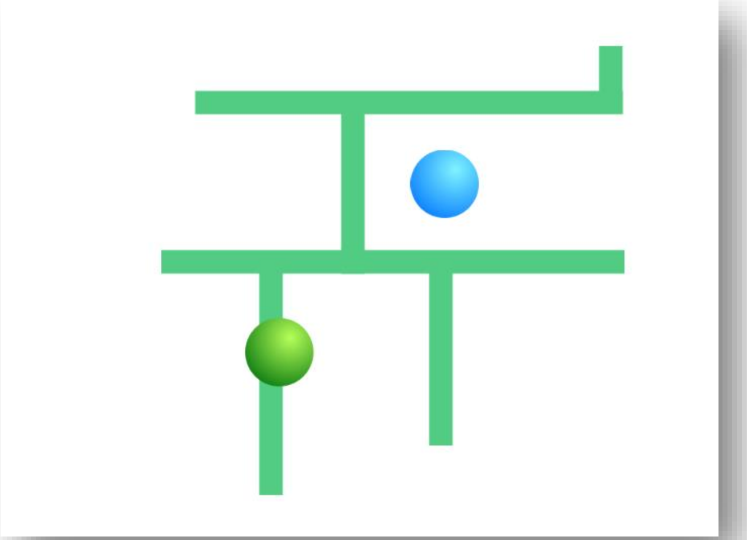
Now all we need to do is make the player sprite go back to start of the maze if it touches the walls. To do this, we need to bring back the “go to” block we used at the start to position our sprite in the bottom left corner.



```
when ⚑ clicked
if < touching Maze ▾ ? > then
    go to x: -150 y: -100
```

Now if the player touches the maze, it will be moved back to the start of the maze, right?

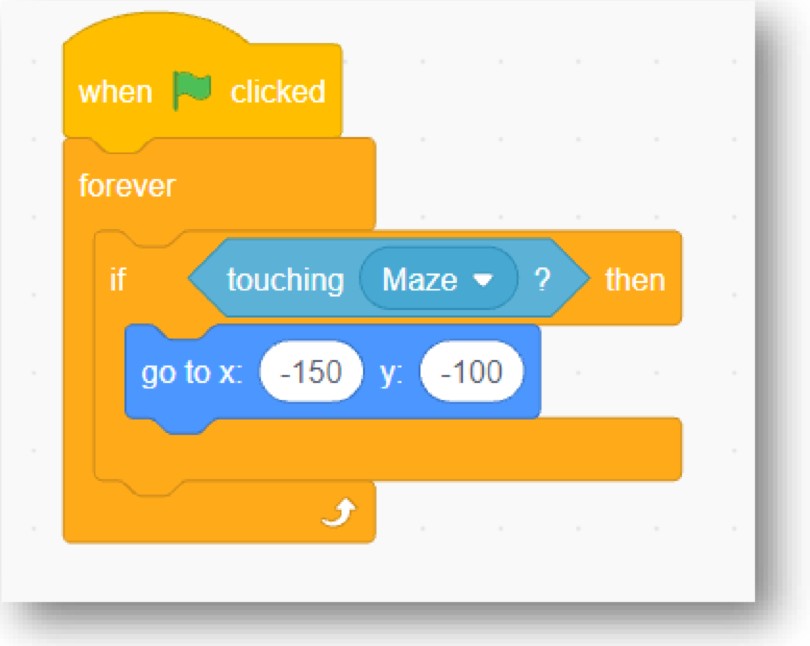
WRONG!



For some reason, it seems our player is still able to touch walls without any issues…

Collisions (the mistake)

While our code did work, there is one small issue. Whether the player is touching the maze or not is only checked once, right when the flag is clicked to start the project. We want this condition to be checked constantly. “Forever” block to the rescue!

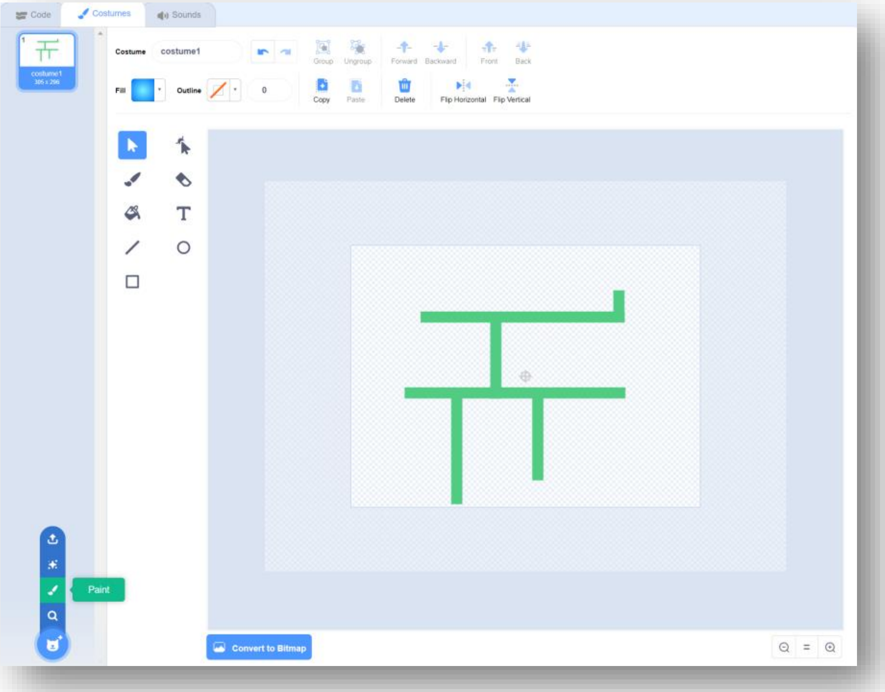


```
when ⚑ clicked
forever
    if touching Maze ▾ ? then
        go to x: -150 y: -100
↻
```

By putting a “forever” block around the “if then” block, this condition will constantly be checked, finally giving us success! Click the flag again, and see what happens when you touch a wall.

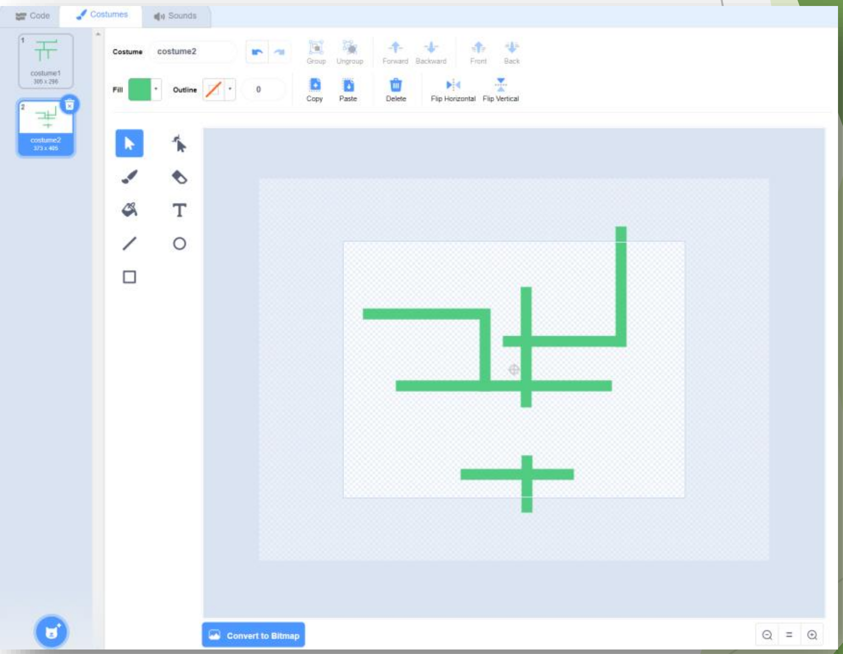
More levels

Before we begin work on our player's collision with the goal, we first need to make more mazes for our game. This is because part of the goal's function will be to go to the next level once touching the player.



Costume costume1

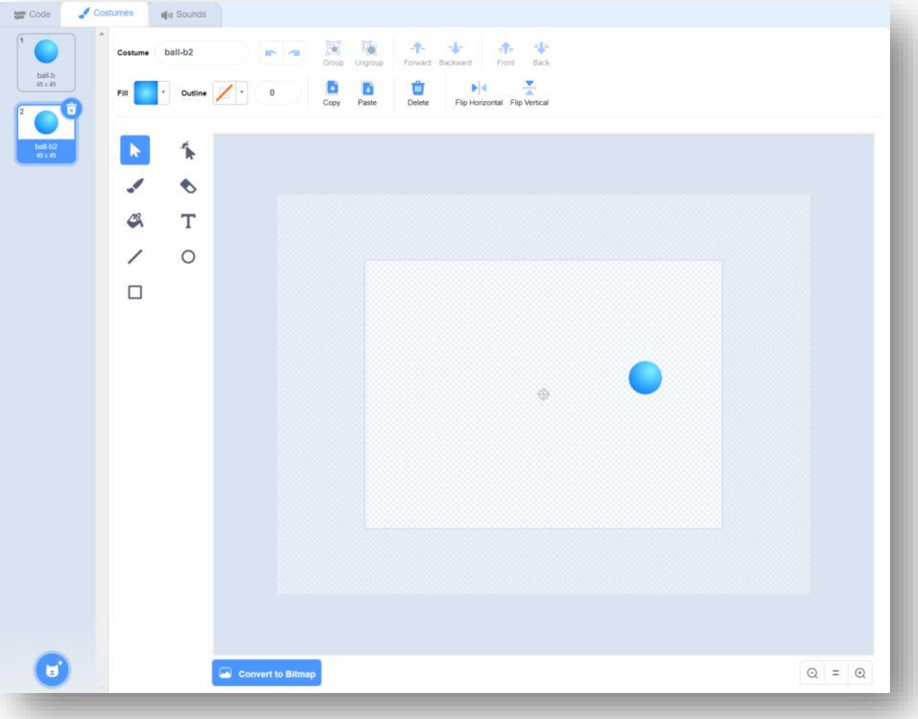
New level design



Costume costume2

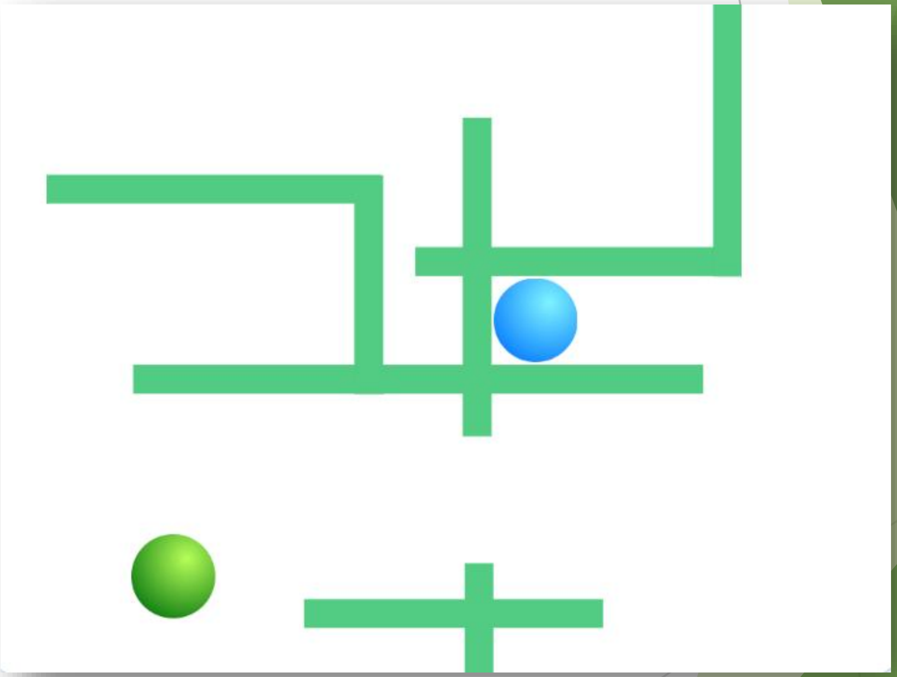
More levels

Remember to duplicate your first costume for your goal sprite, and move it to an appropriate location for your next maze.



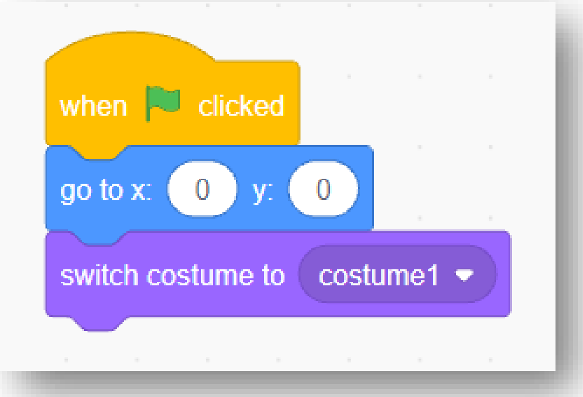
Costume ball-b2

Outline 0



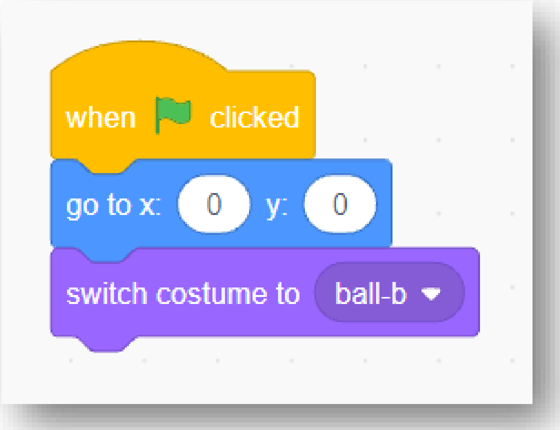
Starting at the right costume

Now that we have multiple costumes for our maze sprite and goal sprite, we need to make sure our game starts at the first of both.



```
when 🏴 clicked
go to x: 0 y: 0
switch costume to costume1 ▾
```

Let’s make sure our maze will look the same in the art editor as on screen by putting it to 0, 0. Then we will use the “switch costume” block from the “looks” category to switch to the first costume

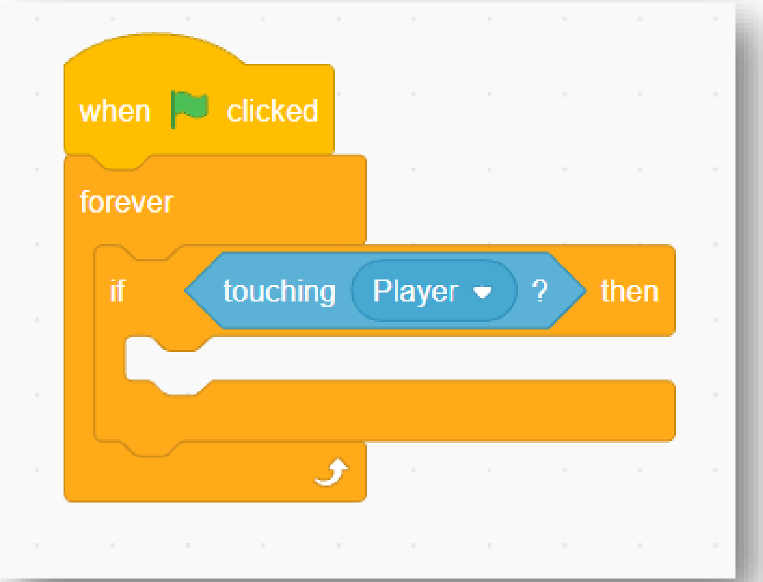


```
when 🏴 clicked
go to x: 0 y: 0
switch costume to ball-b ▾
```

We will do the same for our goal sprite.

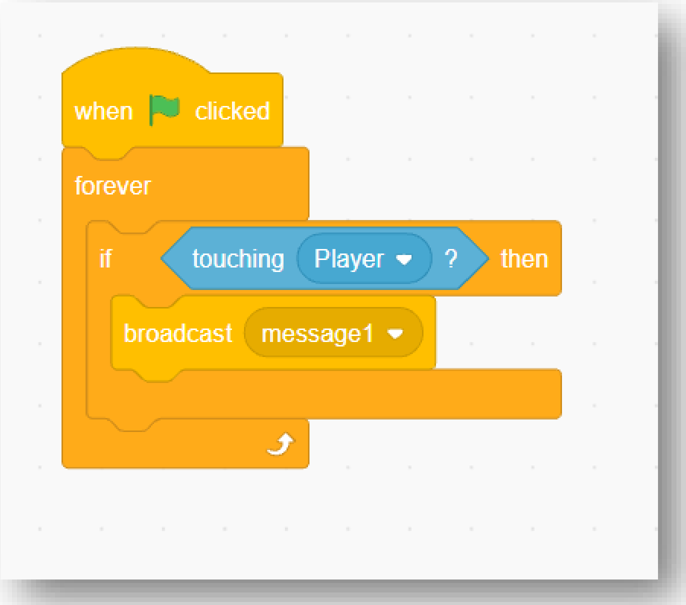
Advancing to the next level

Now we need the level to change when the player touches the goal sprite. When the level changes, we will need 3 things: 1. The player needs to move back to the start of the maze. 2. The maze sprite must go to the next costume. 3. The goal sprite must go to the next costume.



```
when 🏁 clicked
forever
    if touching Player ▾ ? then
```

Let's make this script in our goal sprite, in the same way as we did before for our player's wall collisions.

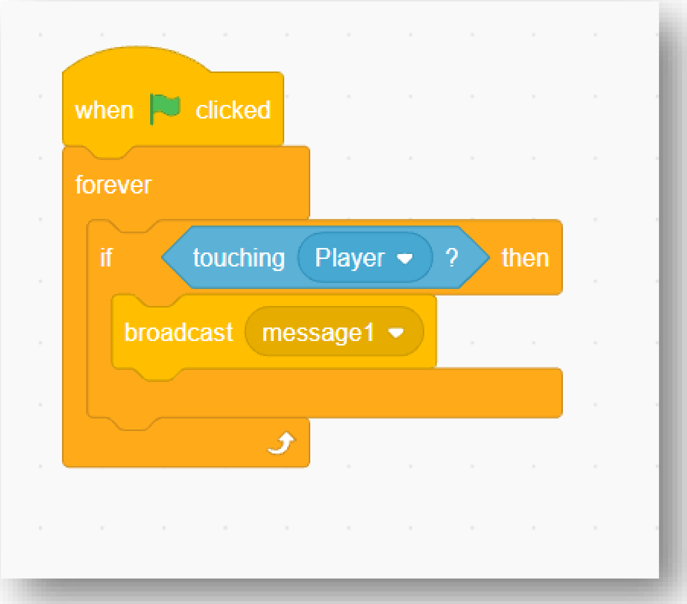


```
when 🏁 clicked
forever
    if touching Player ▾ ? then
        broadcast message1 ▾
```

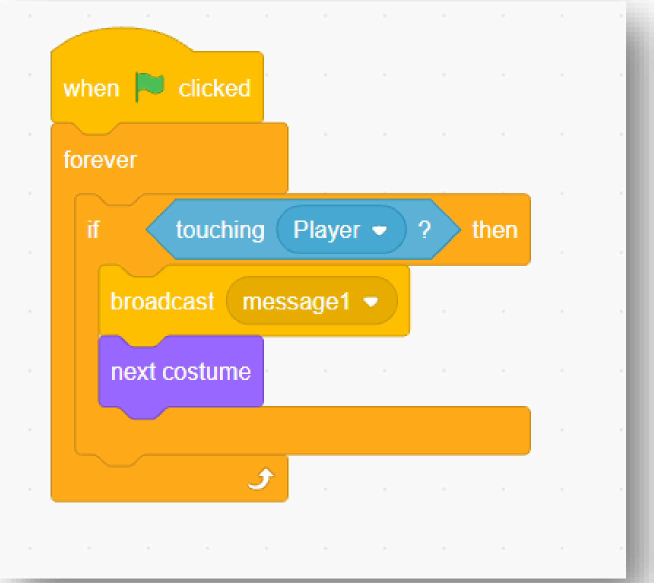
The "broadcast message" block will allow us to communicate with other sprites, and tell them what to do

Advancing to the next level (2)

In addition to just sending a message to other sprites, we also need to change the costume of the goal sprite too, using the “next costume” block, found in the “looks” category.



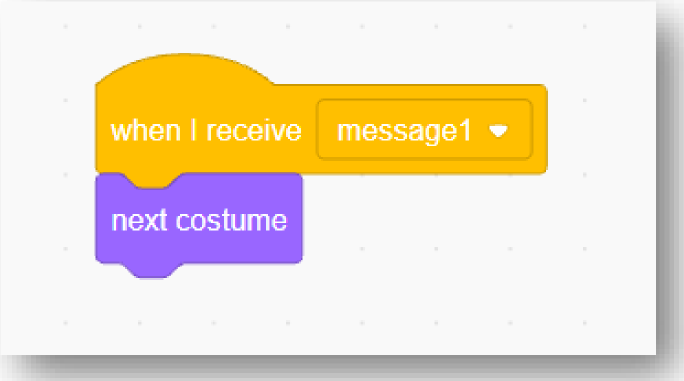
```
when 🟩 clicked
forever
    if touching Player ▾ ? then
        broadcast message1 ▾
    ⤴
```



```
when 🟩 clicked
forever
    if touching Player ▾ ? then
        broadcast message1 ▾
        next costume
    ⤴
```

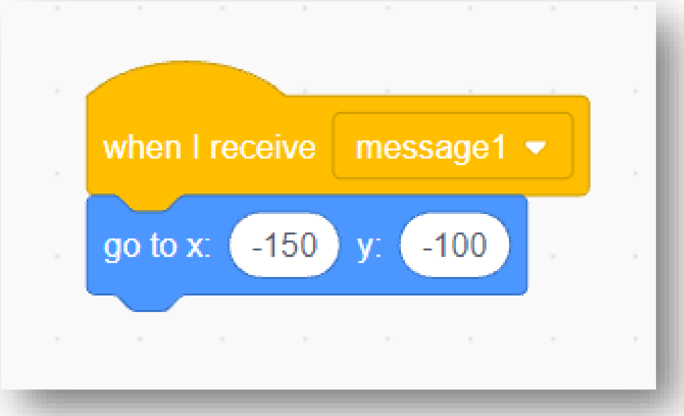
Advancing to the next level (3)

Now let’s make this message affect our other sprites. Use the “when I receive message” block from the “events” category to start the script



```
when I receive [message1 ▾]
next costume
```

Our maze sprite just needs to move to the next costume when it receives the message.

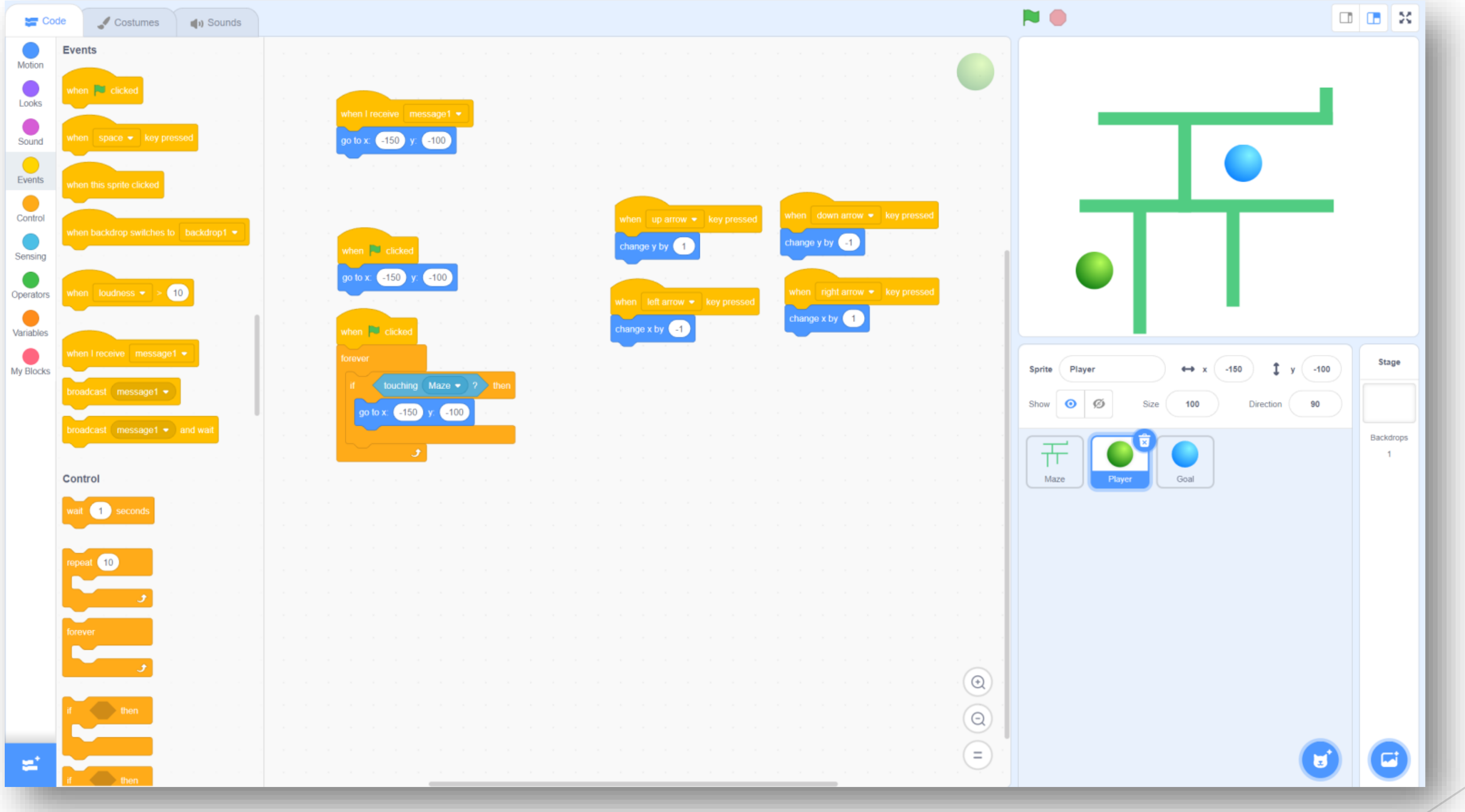


```
when I receive [message1 ▾]
go to x: (-150) y: (-100)
```

Our player sprite just needs to return to the start of the maze when it receives the message

We’re done!

And with that, we’re done! We now have a maze game where we use the arrow keys to move our player to a goal while avoiding walls which send you back to the start of the maze. We’ve also created 2 levels, and you can create as many as you want!



```
when I receive message1
go to x: -150 y: -100

when 🏴 clicked
go to x: -150 y: -100

when 🏴 clicked
forever
    if touching Maze ? then
        go to x: -150 y: -100

when up arrow key pressed
change y by 1

when down arrow key pressed
change y by -1

when left arrow key pressed
change x by -1

when right arrow key pressed
change x by 1
```

Sprite Player x -150 y -100
Show Size 100 Direction 90

Congratulations!

Thanks for Listening!

And happy scratching!