



GAMES FOR CHANGE
STUDENT CHALLENGE

INTRO TO GAME DESIGN

Computer
Programming
and Game
Design with
Scratch



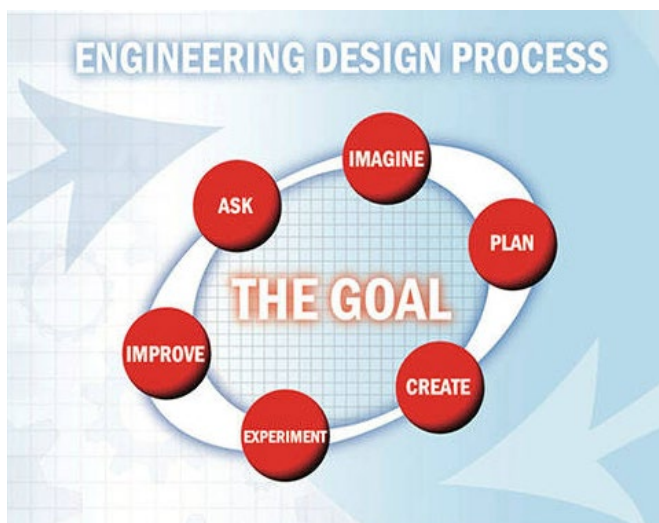
What is Computer Programming?

Computer programming is the process of writing instructions to be carried out by a computer.

Your computer and all of the websites you visit, your phone and all the apps you use, and any video games you play are all made possible by computer programming.

Computer programming is also the driving force behind many of the world's current technological advancements, including artificial intelligence, autonomous vehicles, and virtual reality.

Just like a game designer, computer programmers constantly move through a cycle of prototyping, testing, and tinkering with their programs. This is often called the **engineering design process**. This process is used throughout engineering, science, and design-related fields and exists in several different versions. Here are a version from NASA as well as the Game Design Cycle you used in earlier:

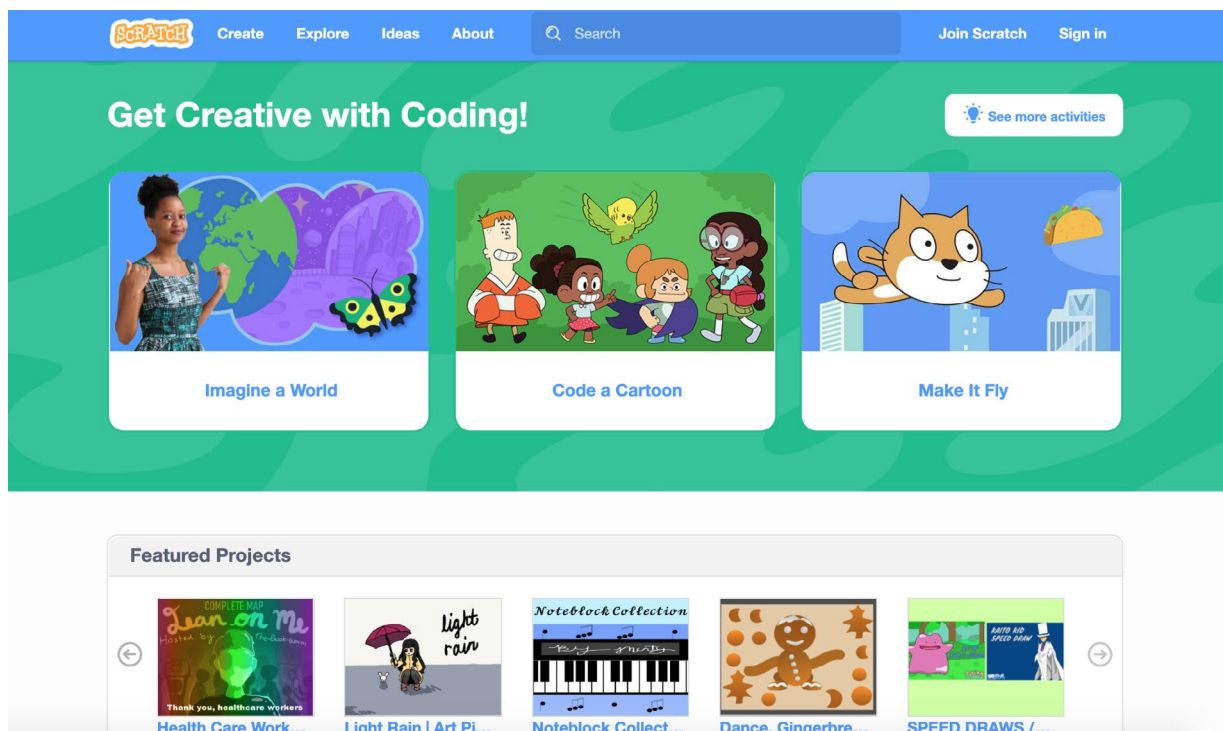


What is Scratch?

There are many different programming languages. Some of the most popular programming languages include Python, JavaScript, and C++. We are going to use a language called Scratch to learn the basics of programming and game design.

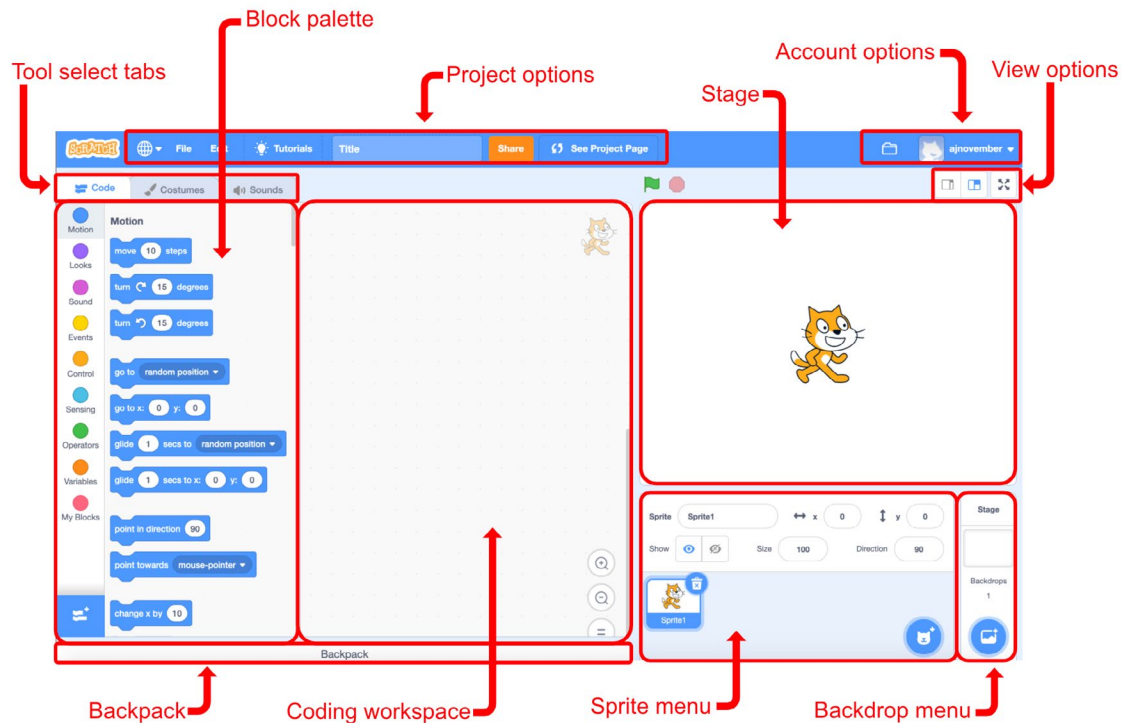
Scratch is a toolkit that combines computer programming with graphic design tools to allow users to create games, animations, and other interactive programs.

Go to Scratch at scratch.mit.edu. Click on “**Create**” in the top-left corner to access the project editor.



The Scratch Project Editor

The Scratch project editor is where all of our coding, designing, and creating will take place.



Stage - Here you will see the output of your project as you're building it

Sprite Menu - Sprites are all characters and objects in our projects. Here you can control what sprites are in your project, and the name, position, visibility, and size of those sprites. Add more sprites by clicking on the blue circle with the cat face in the bottom-right of the menu.

Backdrop Menu - Backdrops are backgrounds for our projects. Add and manage backgrounds here. Add a new backdrop by clicking on the blue circle with the mountains at the bottom of the menu.

Tool Select Tabs - Choose between Code, Costumes, and Sounds tools. Switching between these tabs will change what tools are available in the sections below.

Block Palette - Here you can see all of the available code blocks. Code blocks are separated into color-coded categories, like Motion, Events, Sound etc...

Coding workspace - Here you can write your programs by connecting blocks from the block palette. Note that each sprite will have its own programs.

View Options - Here you can switch between different view modes for the project editor.

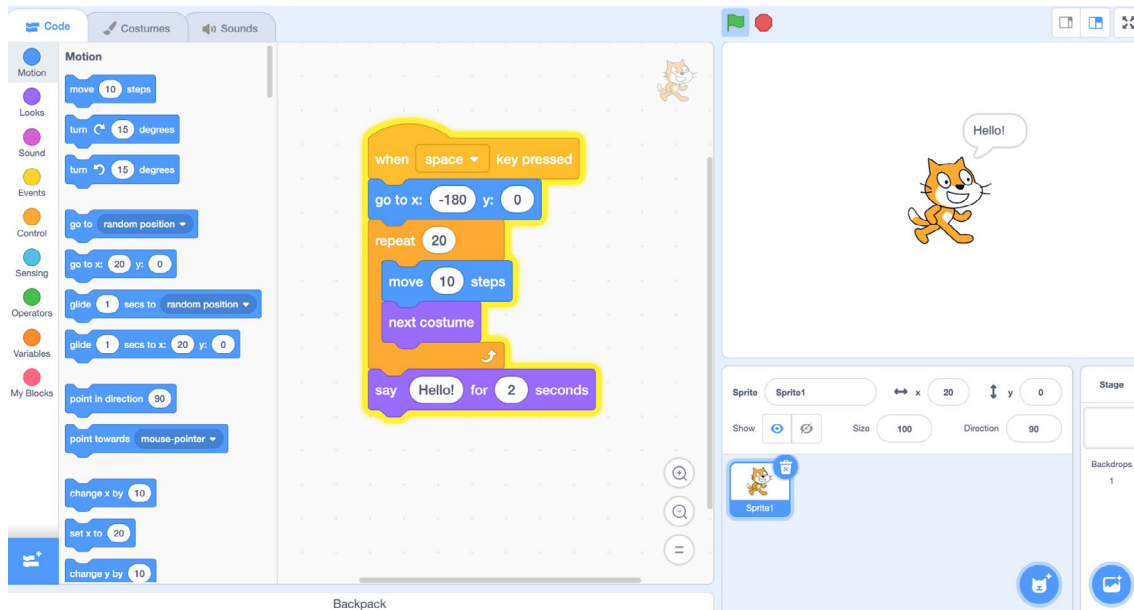
Backpack - The backpack allows you to transfer code, sprites and backdrops between Scratch projects.

Project Options - Here you can save your project, add a title, share your project (which makes it public), change the language, and other options.

Account Options - Here you can access your other projects and other account settings.

Code Blocks

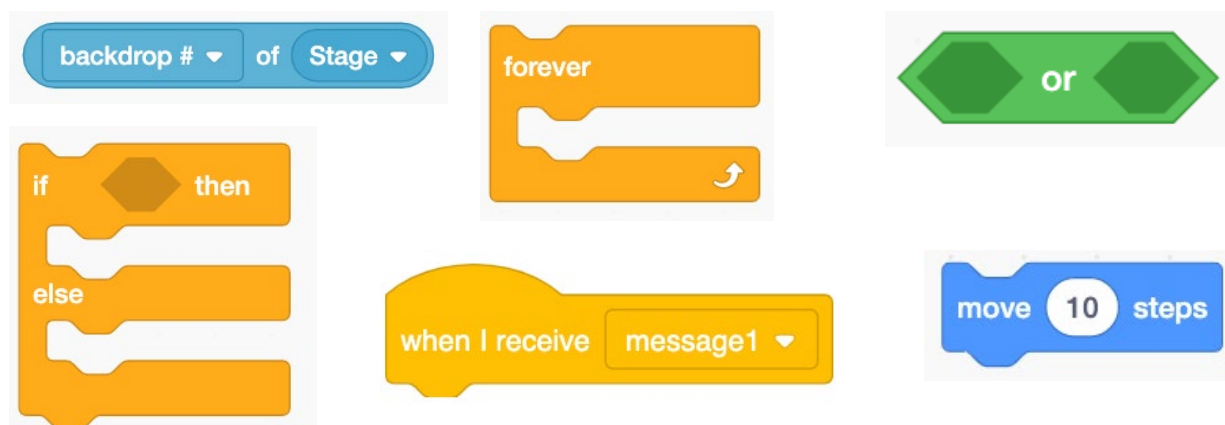
In Scratch, programs are written by connecting the colored code blocks. To start building programs, drag blocks from the block palette into the coding workspace in the center of the project editor..



There are currently 119 different blocks in Scratch, divided into 8 categories: Motion, Looks, Sound, Events, Control, Sensing, Operators, and Variables. We will introduce blocks as they come up in our code, but it is very helpful to spend some time looking through the categories and familiarizing yourself with the different available blocks. Understanding the different tools at your disposal is crucial to being an effective programmer! Visit the [Scratch Wiki blocks guide](#) for detailed information about all of the different types of blocks and how they work together.

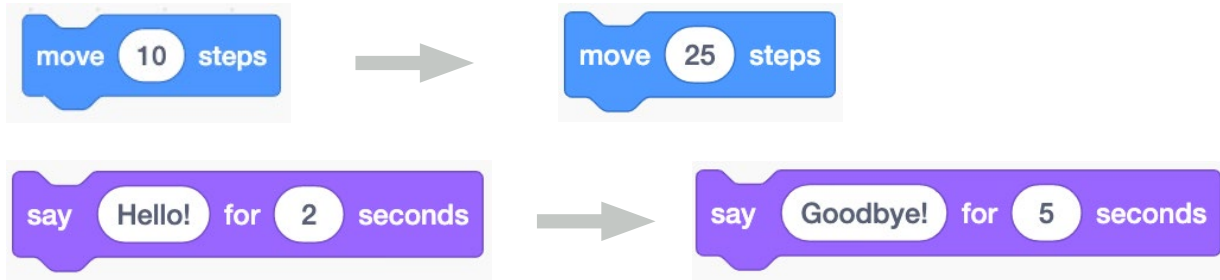
Here are some general tips for working with Scratch's code blocks:

Block shapes - Pay attention to the shapes of blocks and different blocks connect to one another. A block's shape will tell you about how it is to be used in a program and how it will work with other blocks.

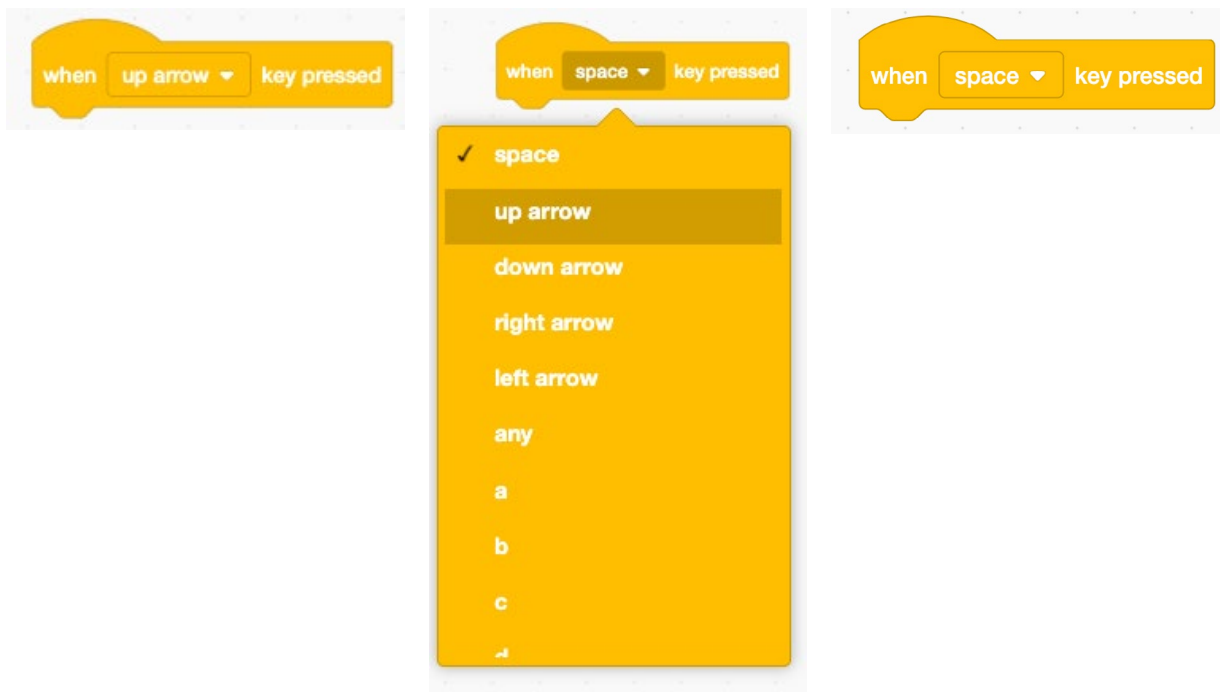


Code Blocks (CONTINUED)

Numbers and Text - In any blocks containing numbers or text inside a white space, these numbers or text can be edited to any value you want! Just click on the text inside the block, delete it, and write whatever you want in its place.



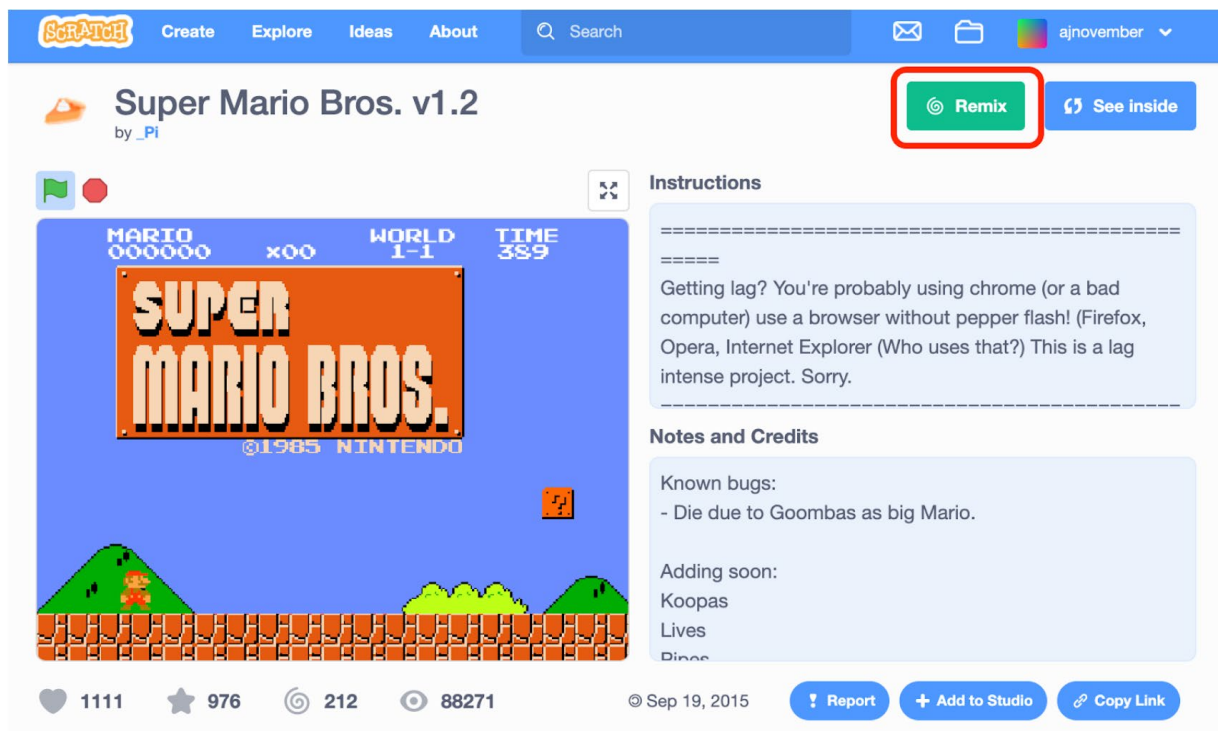
Dropdown Options - Some blocks contain a small white dropdown arrow. Additional options can be selected for these blocks. Just click on the arrow to expose the menu, and select your option.



Remixing

Scratch, in addition to being a programming language and toolkit, is a community of programmers and creators. You can view and play with projects from other users by going to the Scratch homepage and clicking “Explore”. One key feature in Scratch is the ability to Remix others’ projects. **Remixing allows you to add to or change another user’s project.** When you remix another user’s project, you save a copy of the project to your own account, so changes you make will not change the original project.

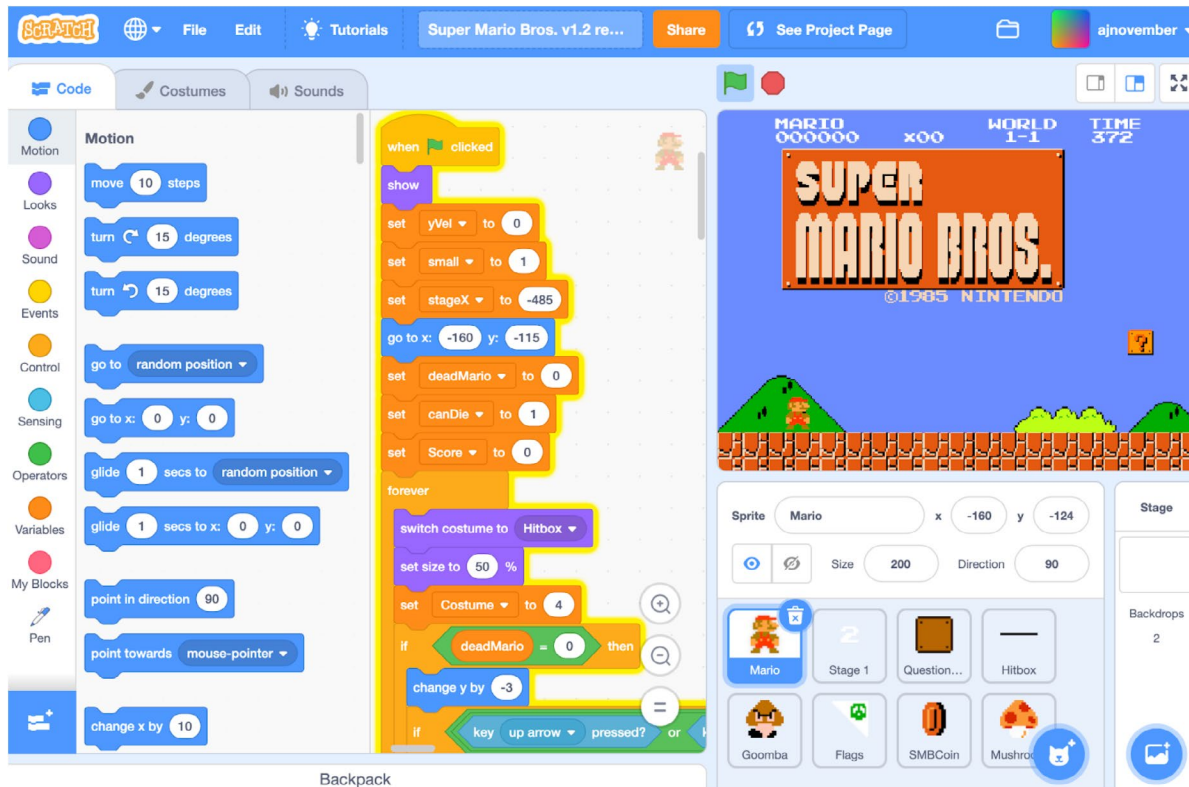
To remix another user’s project, go to the project page and click “Remix”.



This will save a copy of the project to your account and open it in the project editor.

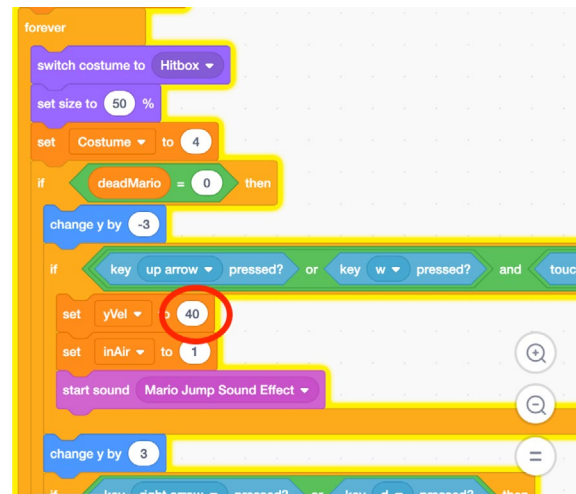
Remixing (CONTINUED)

Remixing and looking inside other's projects is a great way to learn new programming and game design techniques!



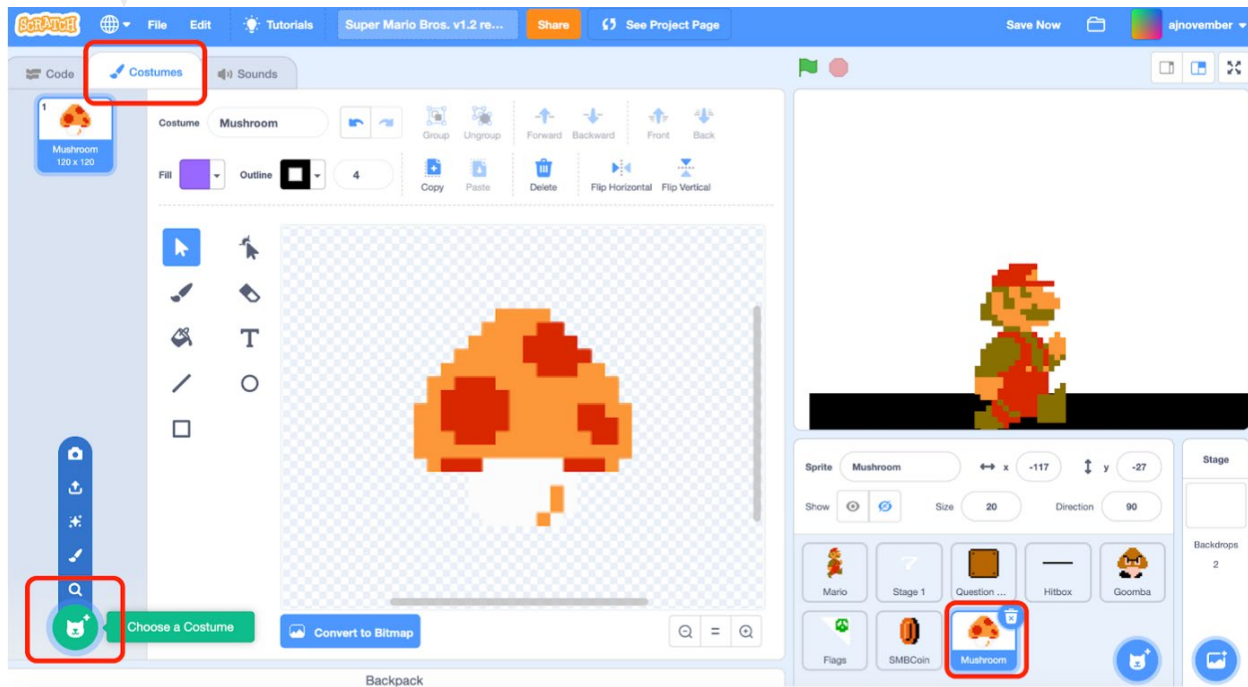
Tips for Remixing Scratch Projects

Tinker Away: It's easy to be overwhelmed when first looking at the code in another user's project. As you begin reading through a project's code, **one easy way to start feeling understanding the code is to change values and numbers that appear in different blocks and see how they affect the game.** Just remember to keep track of what you're changing in case it breaks the game.

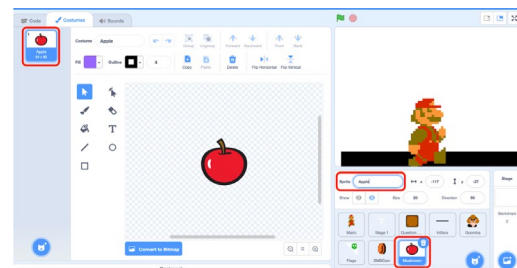
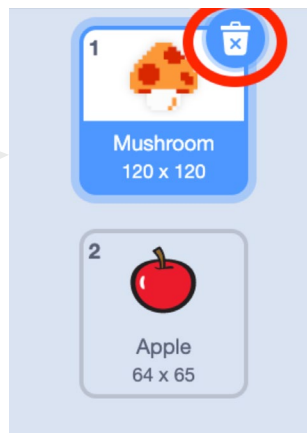


Remixing (CONTINUED)

Changing Sprites: If you want to change a sprite without changing its code, do this by adding a new costume to the sprite. To do this, click on the sprite in the sprite menu, then click on the “Costumes” tab, and on the “Choose a Costume” button in the bottom left. Choose the sprite you want to replace your current sprite.



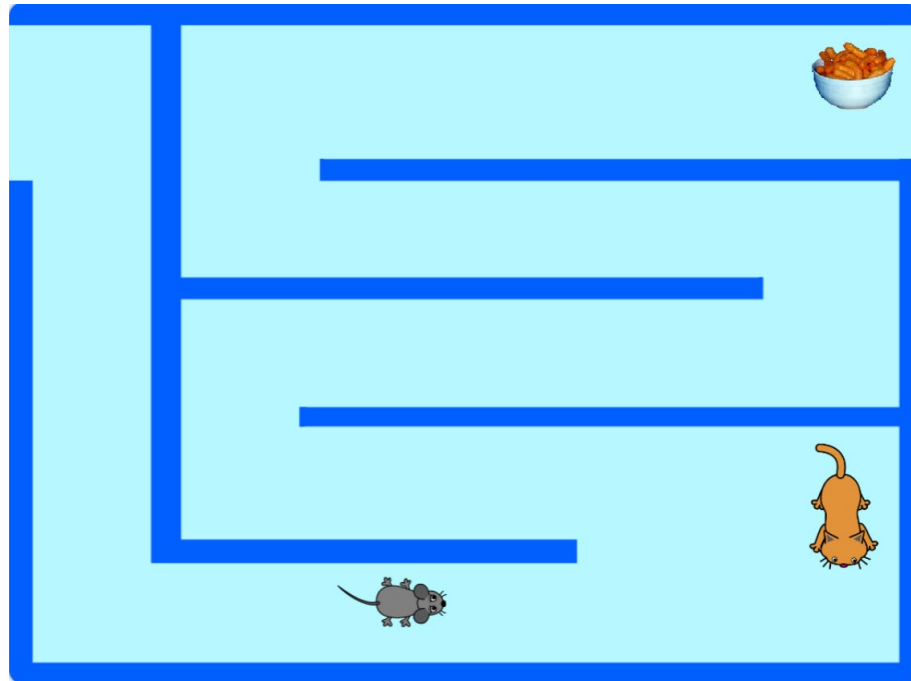
After you pick a new costume, you can delete the original costume by selecting the costume and clicking on the trash can in the top-right corner of the costume.



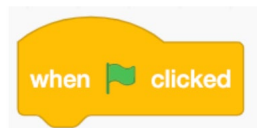
You can also re-name your sprite in the sprite menu.

Project #1: Dungeon Maze

For our first project, we will build a dungeon maze game! You can see (and remix) my version [here](#).

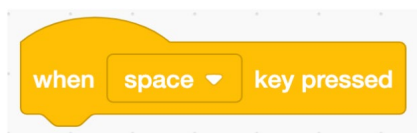


New Blocks Used



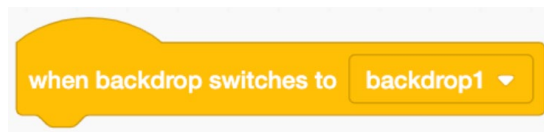
From events

Triggers any attached blocks when the green flag above the stage is clicked



From events

Triggers any attached blocks when the specified key is pressed.



From events

Triggers any attached blocks when the backdrop is switched to the specified one

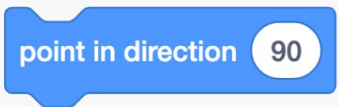
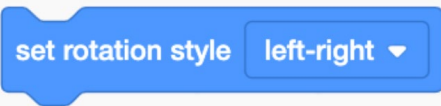

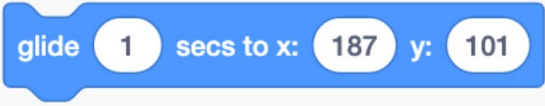

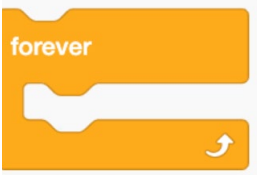

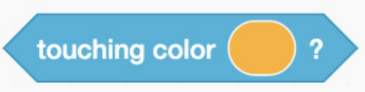


From motion

Moves the sprite the specified number of steps (pixels)

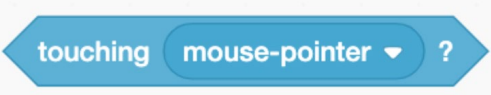

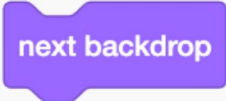
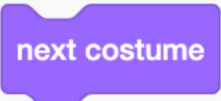


Project #1: Dungeon Maze (CONTINUED)

New Blocks Used (CONTINUED)

	From motion Points the sprite in the specified direction, where 90 degrees points to the right
	From motion Sets the rotation style of the sprite, which determines how a sprite will rotate and flip
	From motion Send the sprite to a specific X, Y position on the stage
	From motion Make a sprite glide to a specific X, Y position in the specified amount of time
	From control Checks a condition from the top of the block. If the condition is true, the blocks inside it will activate.
	From control Repeats any inside blocks forever
	From control Stops all or some pieces of code from running
	From sensing Checks if sprite is touching a specific color

Project #1: Dungeon Maze (CONTINUED)

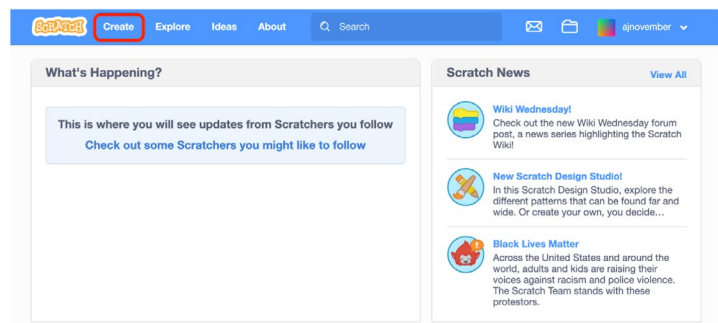
New Blocks Used (CONTINUED)

	From sensing Checks if sprite is touching the mouse pointer or another sprite
	From operators Checks if either condition is true
	From looks Switches to the next backdrop
	From looks Switches to the next costume
	From looks Makes a sprite invisible
	From looks Makes a sprite visible

Phase 1: Basic Movement

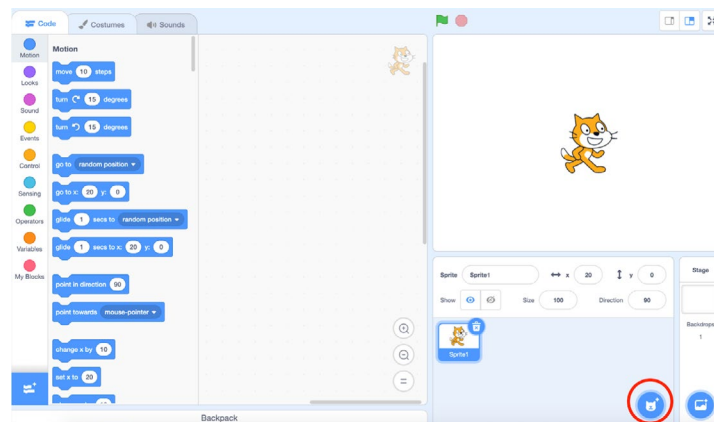
In this first phase, we will begin our project by picking our player sprite and programming basic four directional movement using event and motion blocks.

1. **New project:** Open a new Scratch project by going to scratch.mit.edu, signing into your account, and clicking “Create” in the top left corner.



Project #1: Dungeon Maze (CONTINUED)

2. **Choose your player sprite:** Click on the **“Choose A Sprite”** button in the bottom right corner of the sprite menu. This will launch the sprite library.



For this project we will use the mouse sprite as our player. You can change this to a different sprite later if you like.

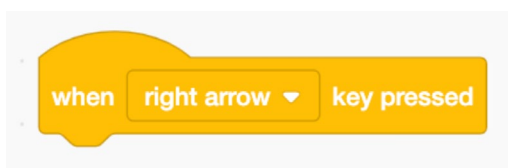


Mouse1

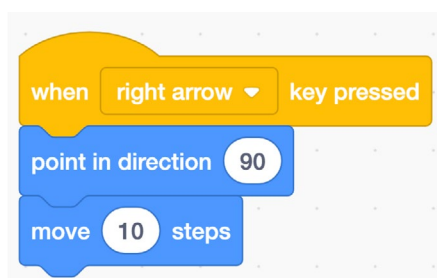
Once you choose your sprite, you can delete the original cat sprite by clicking on the trash can in the top-right corner of the sprite thumbnail. This should leave you with only your new sprite.

3. **Start coding:** Our basic movement script will consist of 3 blocks. Start with the “when space key pressed” block from events. Drag this block into the coding workspace and use the dropdown arrow to modify this block so it uses the right arrow instead of the space key.

Note: In Scratch, our programs will almost always start with a block from the events category. Notice how their curved tops will not allow any blocks to be connected above them.



Next, go to the motion category and connect a **“point in direction 90”** block and a **“move 10 steps”** block.



Project #1: Dungeon Maze (CONTINUED)

4. **All four directions:** Repeat the above block of code for the other three directions.

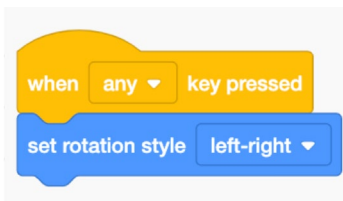
Note: You can copy a chunk of code by right-clicking (control+click) on the top block in the chunk, and selecting “**duplicate**”.

For each direction (right, left, up, down), change the “**when space key pressed**” and “**point in direction 90**” blocks to the correct direction. Make sure your numbers/directions match the combinations below:



Now you should be able to move your sprite in all four directions! Try this using your arrow keys.

If moving your sprite left causes your sprite to flip upside down, add the following code to keep your sprite from flipping upside down: →



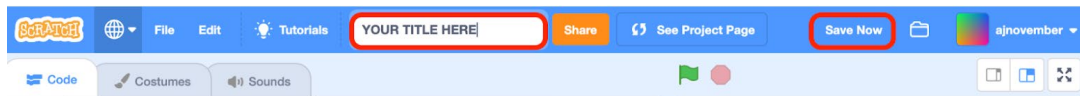
This will restrict your sprite to only rotate horizontally, not vertically.

5. **Bonus:** Click on the “**Costumes**” tab near the top left of the screen. If your sprite has multiple costumes that can be used to show a walking animation, try adding a “**next costume**” block from the looks category to the end each of your four directions. This will only work well for specific sprites. →



Project #1: Dungeon Maze (CONTINUED)

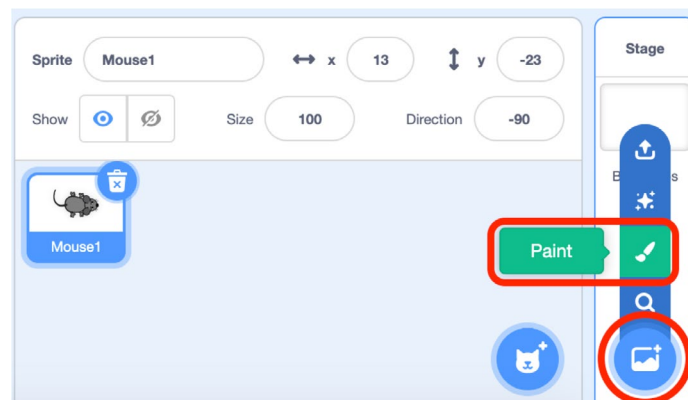
6. **Title and save:** On the top of the screen, add a title to your project. Then look for the words “Save Now” next to the picture of a folder near the top-right corner of the screen. If you do not see these words, that means your project has already been saved!



Phase 2: Create a Backdrop

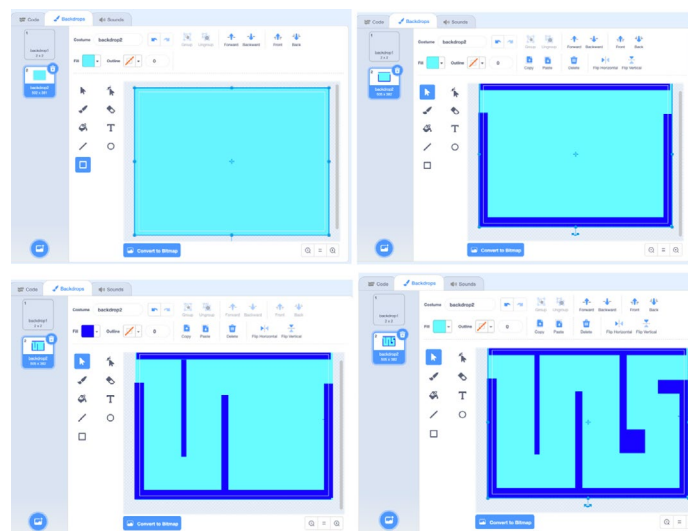
Now that we’ve programmed basic four directional movement for our sprite, it’s time to build our maze. We will do this by painting a new backdrop.

1. **Open the paint tool:** Hover over the “Choose a Backdrop” button on the bottom-right corner of the screen and click on the paintbrush icon to begin painting a new backdrop.



2. **Paint your maze:** Create a simple maze for your game. Keep it simple! We will make more levels later that can be more difficult. Be sure to leave a wide path and a clear entrance and exit for your maze.

Use the **rectangle** and **line** tools to create shapes, and then the cursor tool to adjust the size and position of the shapes.

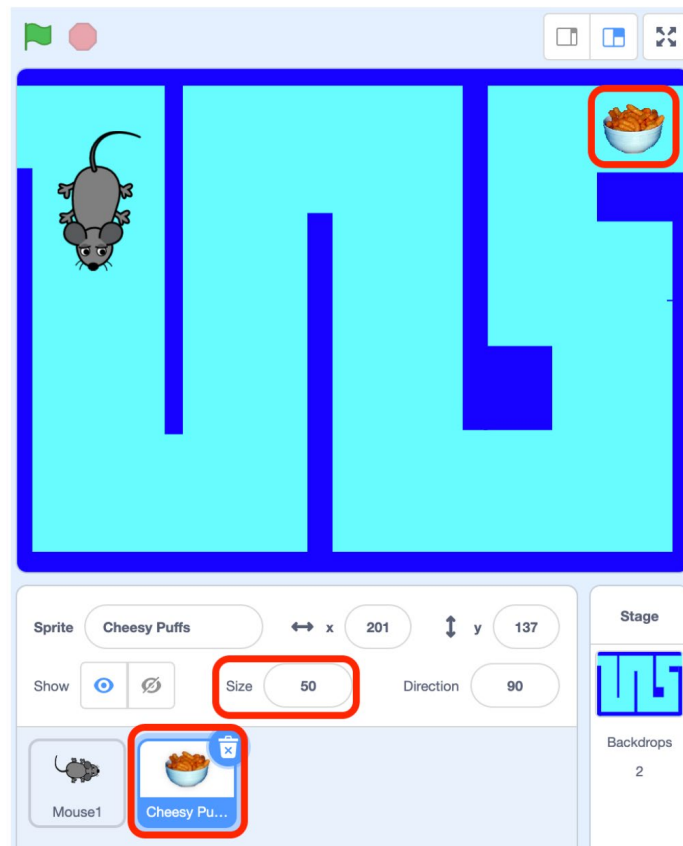


Project #1: Dungeon Maze (CONTINUED)

- Choose a goal:** Choose a sprite to be the goal at the end of the maze. Click on the **“Choose A Sprite”** button on the bottom-right of the screen and choose a sprite from the library. For now, we will use the **Cheesy Puffs** sprite for this.



Place your cheesy puffs at the end of your maze. Make them smaller by finding the size field in the sprite menu and changing the number to 50.



- Save your project.** In the top-left of your screen, click **File > Save Now** just to be sure.

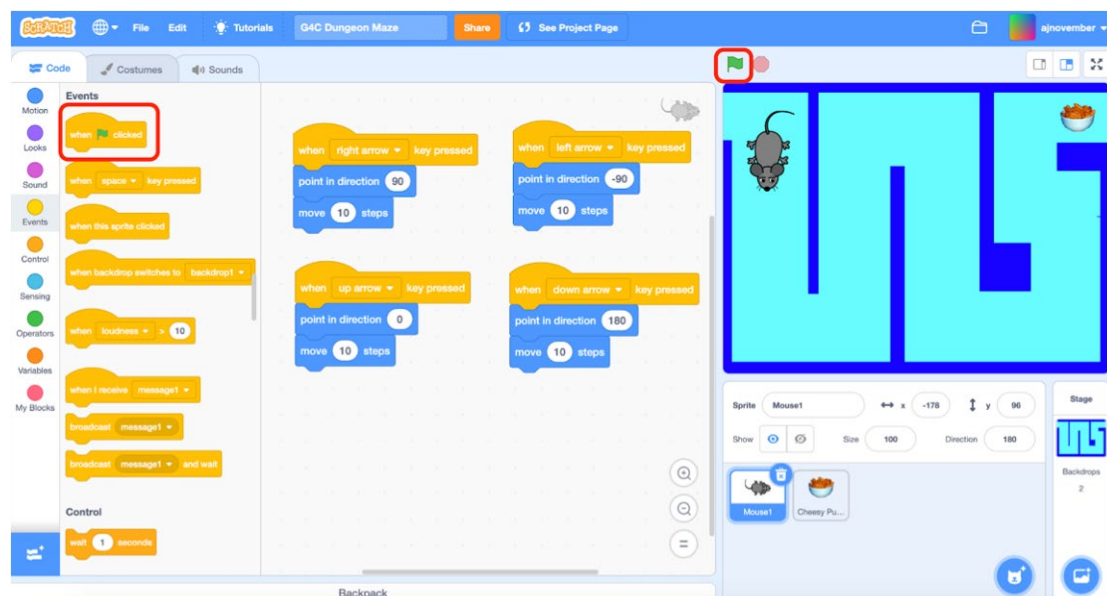
Phase 3: Set the Rules

Now we will program some of the rules of our game. The basic rules will include the following:

- The player starts at the beginning of the maze and must walk to the end of the maze using their arrow keys.
- If the player touches a wall, they must go back to the beginning of the maze.
- When a player reaches the end of the maze, they may continue on to the next level.

Project #1: Dungeon Maze (CONTINUED)

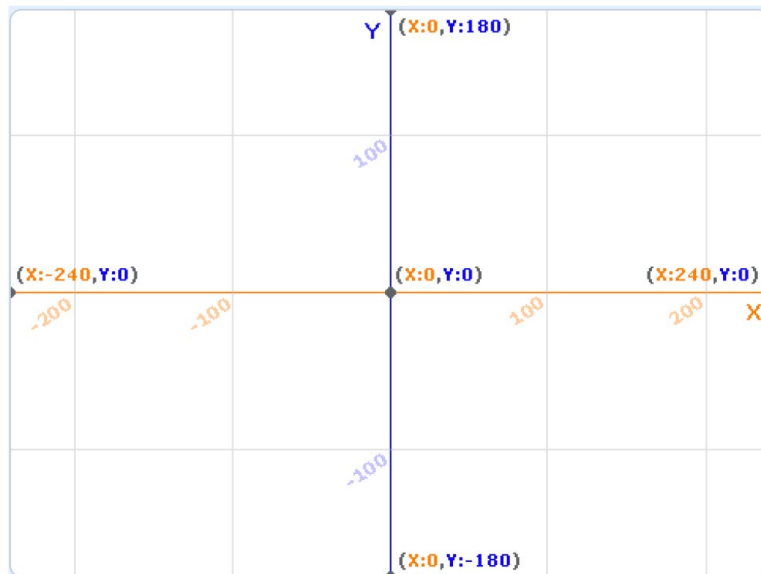
1. **Resize your sprite:** Your mouse is likely too large to make it through your maze. Make your sprite smaller by finding the **size** field and changing the number to 50.



2. **The Green Flag:** The “when green flag clicked” block from **events** is often used as a start button for Scratch games. The green flag in this block refers to the green flag just above the stage. When the green flag above the stage is clicked, any blocks connected to the “when green flag clicked” block will be triggered. The red stop sign next to the green flag can be used to stop your program at any time.

Project #1: Dungeon Maze (CONTINUED)

3. **X and Y coordinates:** To set the starting position for the mouse, we will need to use X and Y coordinates to describe where the sprite should begin. X and Y coordinates are a common way to describe position in any digital design tools you may use. Here is a diagram that shows how X and Y values are measured in Scratch.

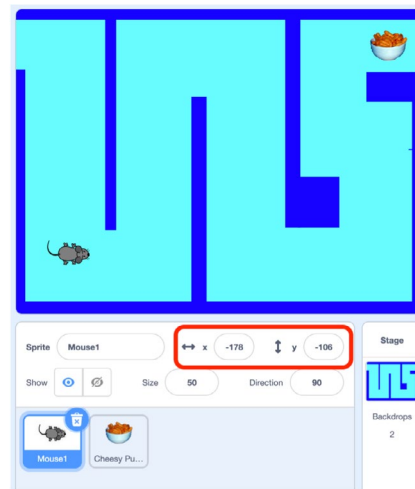


Origin - The origin is the point where both X and Y equal 0. All coordinates will be measured from this point. In Scratch, the origin is in the center of the stage.

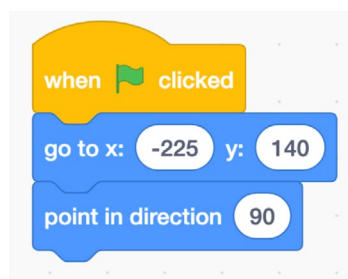
X Coordinate - Describes an object's horizontal position left or right of the origin.

Y Coordinate - Describes an object's vertical position above or below the origin.

Note that you can see the X and Y coordinates of any sprite at any time by selecting that sprite and looking in the X and Y fields in the sprite menu. →



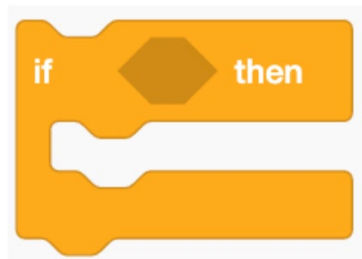
4. **Set starting position:** We will now set the starting position of our player sprite. First, drag your player sprite to the beginning of your maze. This piece of code will start with a “when green flag clicked” block from events. Then, connect a “go to x_y_” block and a “point in direction 90” block from motion.



Press the green flag and see if your sprite goes to the beginning of the maze. If not, place your player sprite at the sprite of the maze, and copy the coordinates from the X and Y fields on the sprite menu.

Project #1: Dungeon Maze (CONTINUED)

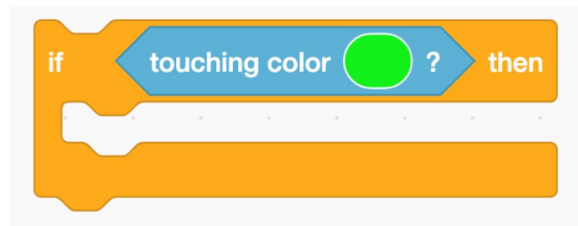
5. **Program the walls:** Right now, our mouse can walk right through the walls. We are going to write a program that makes the mouse return to the beginning of the maze when it touches a wall. To do this, we will use an “if _ then _” block from control. —>



The “if _ then _” block is one of the most important blocks in Scratch, and is a widely used idea throughout other computer programming languages. This block will cause a specific outcome to occur if a condition is met. In this case, our statement will say something like:

“If the mouse touches the wall, then the mouse goes back to the beginning of the maze.”

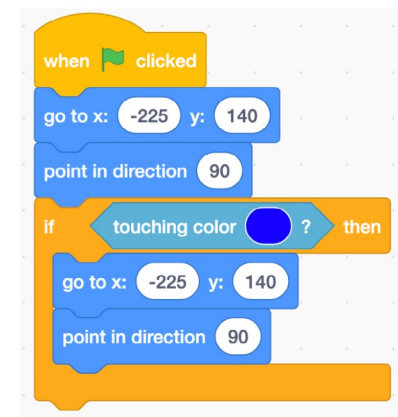
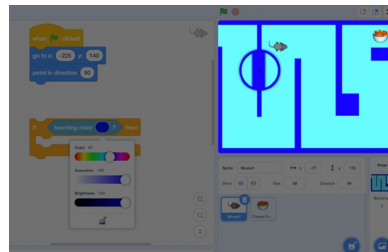
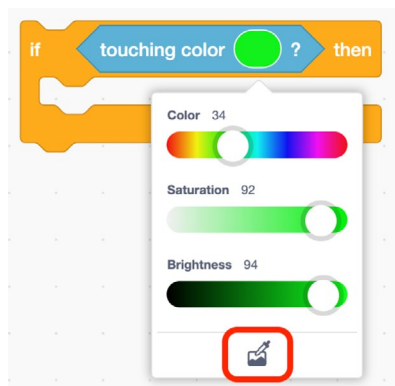
We can achieve this by using the “touching color _” block from sensing. Notice how this block’s shape allows it to fit directly into the hexagon hole at the top of the “if _ then _” block. —>



To set the color we are checking for in the “touching color” block, first click the colored circle inside the block. When the color menu pops up, click the dropper icon at the bottom.

This will darken the entire screen except for the stage. Now bring your cursor to the stage and click on the color that you want to sense. In our case, this is whatever color your walls are.

Now, place another “go to X_ Y_” and “point in direction” block from motion inside the “then” portion of the “if _ then _” block. Make sure the X and Y coordinates match the coordinates of your starting position. You can now add this code to the code that sets the starting position.



Unfortunately, the above code will not fully work yet. As written, the above code will only check to see if we’re touching the wall one time, at the very instant that the green flag is clicked. In order to repeatedly check this, we need to add a loop around our “if _ then _” block. For this, we can use a “forever” block from control.

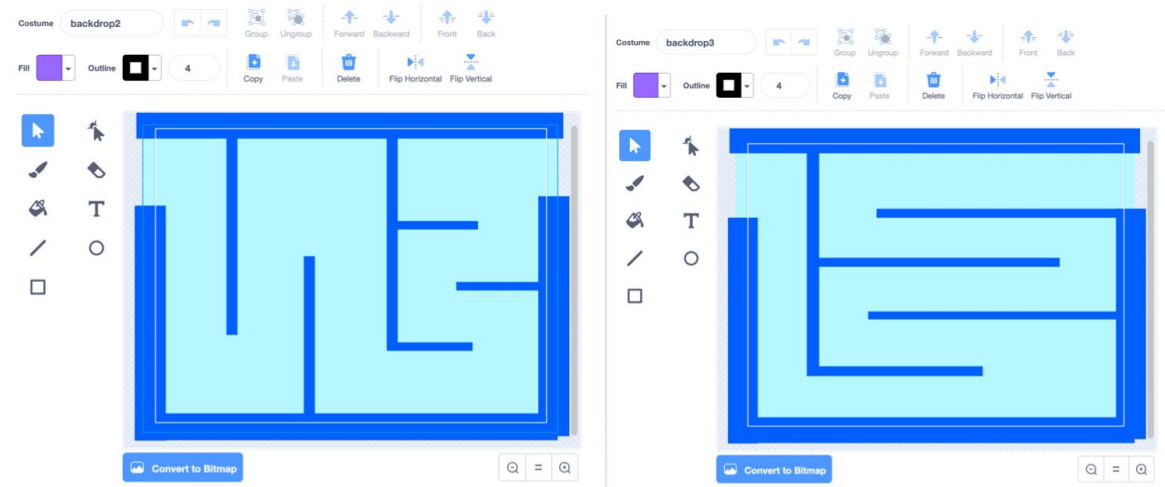
6. **Try it out:** Try running your mouse into a wall. Does it send your sprite back to the beginning like it’s supposed to? **Save your project here** before moving on.

Project #1: Dungeon Maze (CONTINUED)

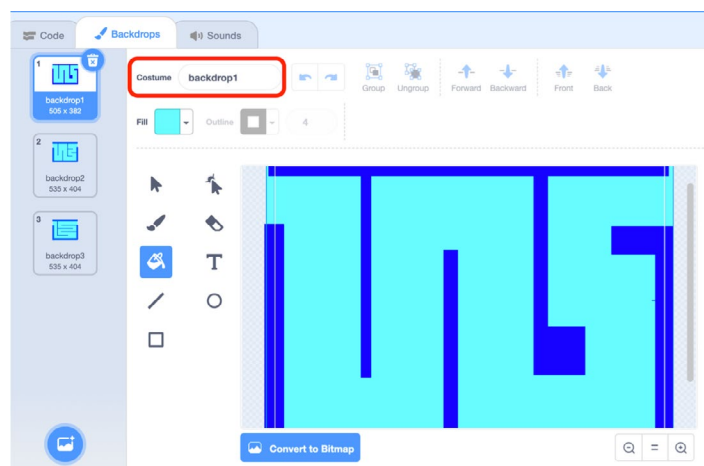
Phase 4: More Levels!

Now we will create more levels for our maze game, and write code that allows us to advance through these levels!

1. **Painting more mazes:** Create two more mazes using the same process that you used to create the original one. Keep your mazes' entrance and exit in the same place. Try to make these ones more challenging, but not impossible!



2. **Backdrop management:** Organize your backdrops so you have three backdrops named "backdrop1", "backdrop2" and "backdrop3", with "backdrop1" being the easiest and "backdrop3" being the hardest. This may involve renaming some backdrops and deleting empty ones.



Project #1: Dungeon Maze (CONTINUED)

3. **Set the goal:** Now we will write code that allows our player to pass to the next level when the mouse reaches the cheesy puffs. First, add another “if_then_” block to our “forever” loop. This time we will use the “touching_” block from sensing as the condition. Use the drop down arrow inside the “touching_” block to select the cheesy puffs.

We need two things to happen when the mouse reaches the cheesy puffs: We need to advance to the next level, and we need to send the mouse back to the beginning of the maze. We will reset the mouse with the same blocks we used earlier, and add an “next backdrop” block from looks to advance to the next backdrop.

```
when clicked
  go to x: -225 y: 140
  point in direction 90
  forever
    if touching color blue? then
      go to x: -225 y: 140
      point in direction 90
```

```
when clicked
  go to x: -225 y: 140
  point in direction 90
  forever
    if touching Cheesy Puffs? then
      go to x: -225 y: 140
      point in direction 90
      next backdrop
```

4. **Set starting backdrop:** Now that we have multiple backdrops, we need to make sure we start on the correct one when the game begins. To do this, go to looks and add a “switch backdrop to backdrop1” block immediately under the “when green flag clicked” block. Use the drop down arrow to select the proper backdrop inside this block if necessary. —>

Test it out! Complete the first level of your maze. Does the second level appear after you beat the first? Can you advance to the third from the second? Now is another good time to **save your project**.

- 5.

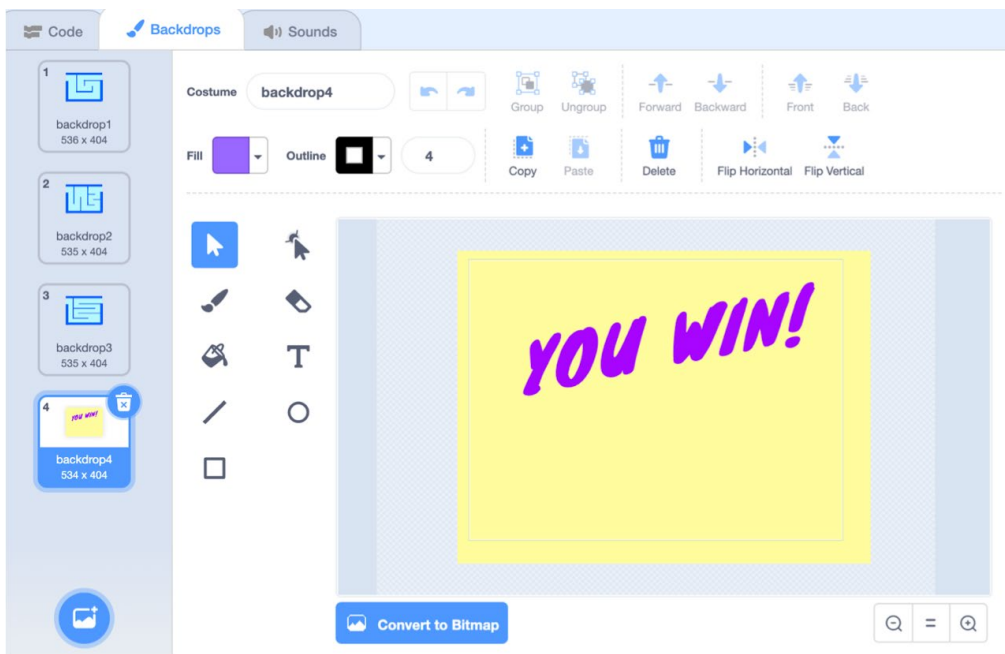
```
when clicked
  switch backdrop to backdrop1
  go to x: -225 y: 140
  point in direction 90
  forever
    if touching color blue? then
      go to x: -225 y: 140
      point in direction 90
    if touching Cheesy Puffs? then
      go to x: -225 y: 140
      point in direction 90
      next backdrop
```

Project #1: Dungeon Maze (CONTINUED)

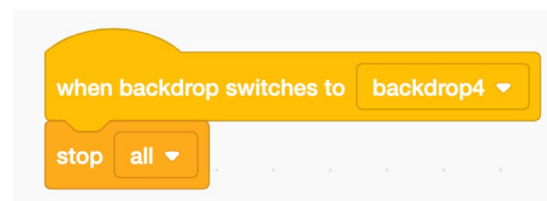
Phase 5: Finishing Touches

Now we will add some finishing touches to our game.

1. **Victory screen:** Create a new backdrop to act as your victory screen! This backdrop will appear when the player has beaten all of the levels. Call it “backdrop4”. Customize this backdrop however you like!



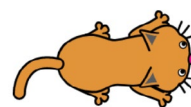
2. **Stop the game:** Add the following two block combination to your code. This will stop everything when you reach the end of the game. “**When backdrop switches to _**” can be found in **events**, and “**stop all**” in **control**. Switch the backdrop referenced in the “**when backdrop switches**” block if necessary.



Phase 6: Bonus Challenge

For a bonus challenge, you can add an enemy to the dungeon maze.

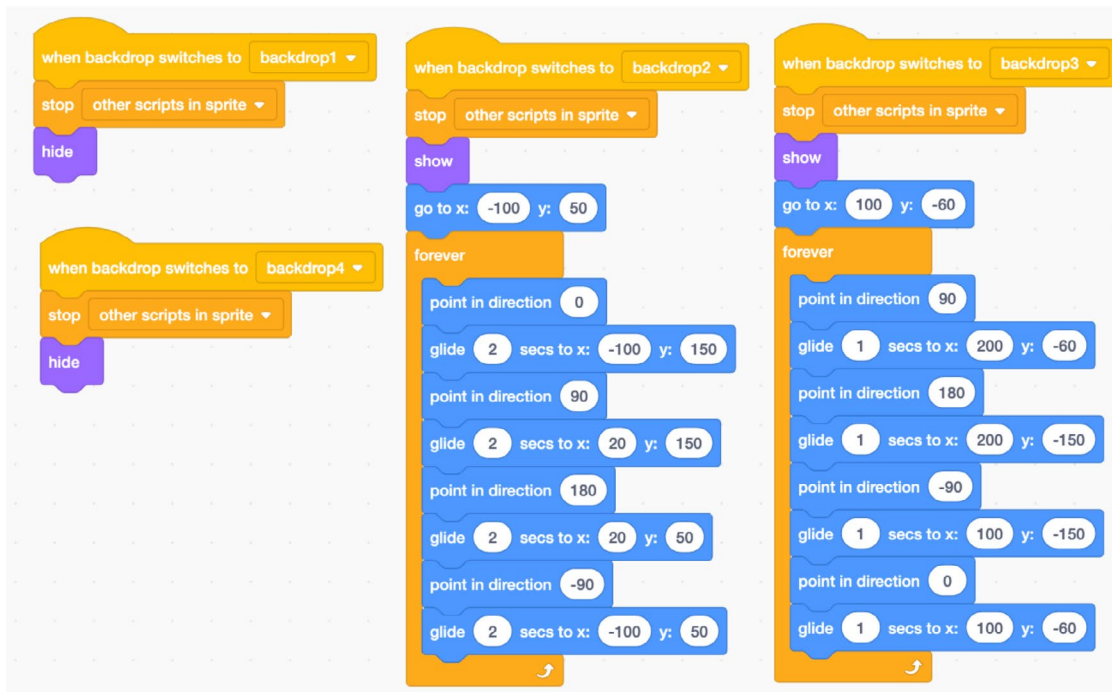
1. **Choose a sprite:** Start by adding a new sprite to be your enemy. I will use the sprite “Cat 2” for this.



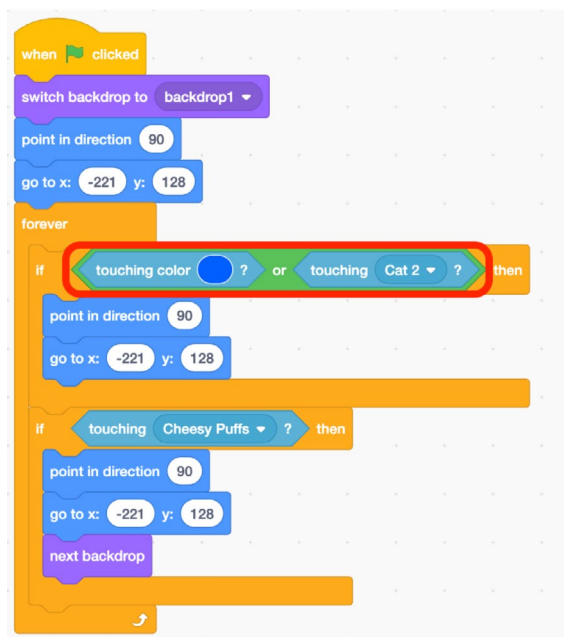
Cat 2

Project #1: Dungeon Maze (CONTINUED)

- Program behavior:** Program behaviors for the enemy sprite. Use the “when backdrop switches to_” block from events to program different behaviors for different backdrops. Here I am using “show” and “hide” blocks from looks to make the sprite only appear on some levels. On levels where the cat appears, I am using “glide _secs to x_ y_” blocks inside of a “forever” loop to program automatic movement for the sprite. You will need to decide where your enemy sprite moves depending on the layout of your maze!



- Add a rule:** Lastly we will modify the rules so running into the cat sends the player back to the beginning. To do this, we will use an “or” block from operators and another “touching_” block to add a second condition to the “if_ then_” blocks that checks to see if we’re touching the walls.



Project #1: Dungeon Maze (CONTINUED)

Playtest

Now that your project is in a playable state, spend some time playtesting it and letting others play it. Here are some questions to guide you while you playtest:

- Does the game work the way you want it to?
- Are there any unexpected behaviors in the game?
- Is the game fun to play?
- Can you actually reach/beat every level?
- Does the game feel too easy, too hard, or just right?
- What could be added to make the game more challenging?
- What could be added to make the game more fun?
- Is there anything confusing or unnecessary that should be removed?

Another way to go about playtesting is to use the acronym **T.A.G.** TAG stands for:

Tell something you like.

Ask a question.

Give a suggestion for feedback.

This can be a helpful prompt for those playtesting and giving feedback on your game, or for you as you playtest other's games.

Customize and Iterate

Take your feedback from playtesting and think of ways to improve and customize your project! Here are some ideas on how to take this project further:

- Change which sprites you are using
- Add more levels to your game
- Add some collectable objects to the game
- Add more enemies or obstacles
- Add a time limit for completing each level
- Improve the detail or design of the backdrops

Project #2: Don't Drop the Donuts

For our second project, we will **make a game where the player must catch falling objects.**

You can see (and remix) my version [here](#).



New Blocks Used



From motion

Changes the Y position of a sprite by the specified amount



From motion

Sets the X position of a sprite



From motion

Sets the Y position of a sprite

Project #2: Don't Drop the Donuts (CONTINUED)

New Blocks Used (CONTINUED)

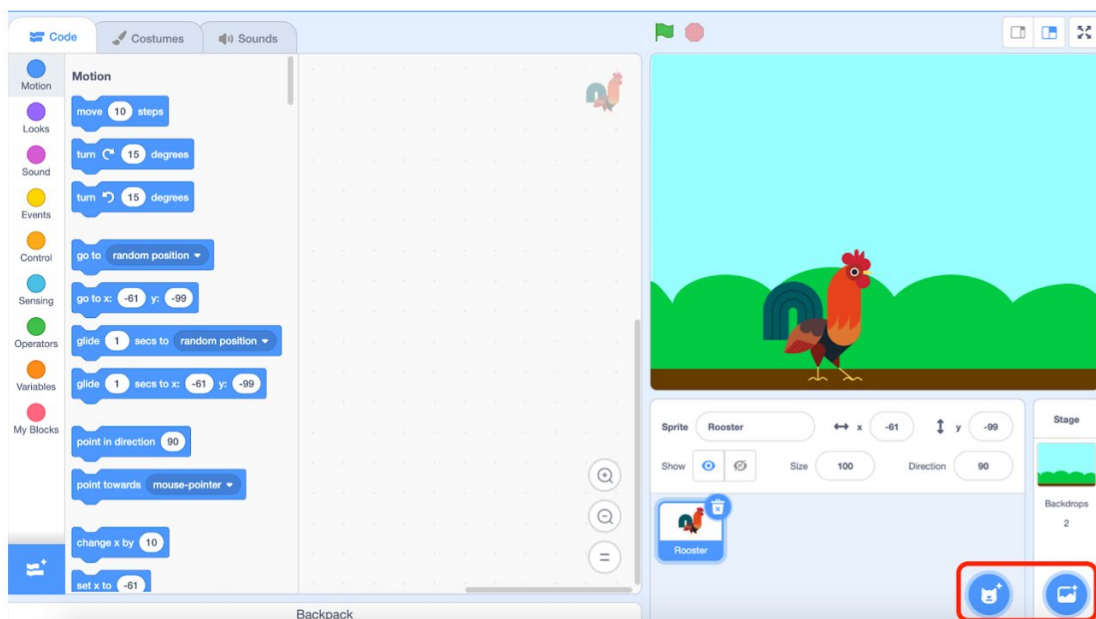
	From motion Rotates a sprite by the specified amount
	From motion A sprite's current Y position
	From operators Compares two numbers. Condition is met if the first number is less than the second.
	From operators Picks a random number between the two specified values.
	From sensing The current X position of the mouse
	From control Waits for the specified amount of time
	From variables Sets the value of a variable
	From variables Changes the value of a variable

Project #2: Don't Drop the Donuts (CONTINUED)

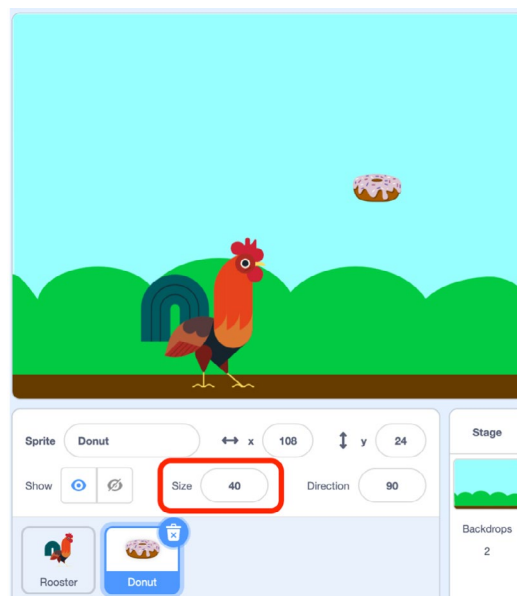
Phase 1: Project Setup

In this first phase, we will open a new project, and pick our backdrop and sprites.

1. **New project:** Open a new Scratch project by going to scratch.mit.edu, signing into your account, and clicking "Create" in the top left corner. Add a player sprite and a backdrop to your project.
 - For the player sprite, any animal, person, or character will work well.
 - For the backdrop, something simple and not too busy will work best.



2. **Falling object:** Add another sprite to be your falling object. I will use the **donut sprite**, but any food or other object will work well. Change the size of your falling object to **40** in the **size field** of the sprite menu.

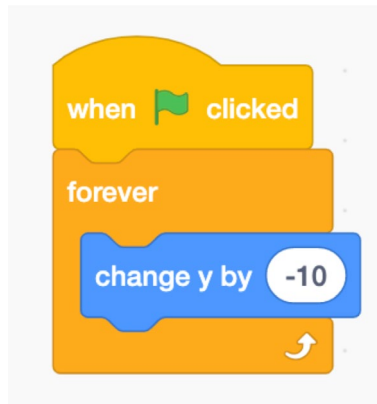


Project #2: Don't Drop the Donuts (CONTINUED)

Phase 2: Program Falling Object

Now we will write code for the falling object.

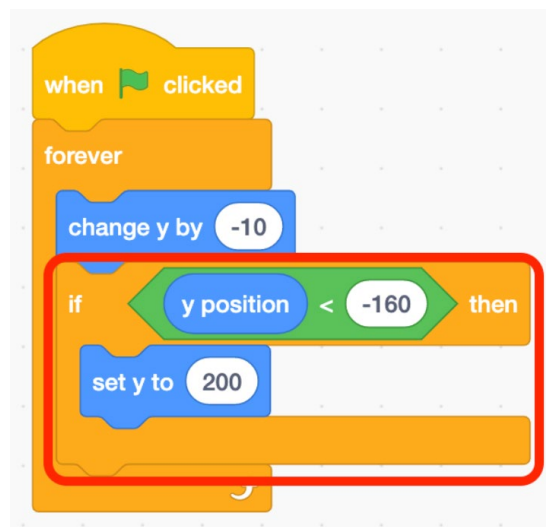
1. **Forever Falling:** Select the falling object sprite and write the following code:



Remember a sprite's **Y coordinate** is its vertical position, so changing Y by -10 will make the sprite appear to fall. Click the green flag to see if the object falls to the ground.

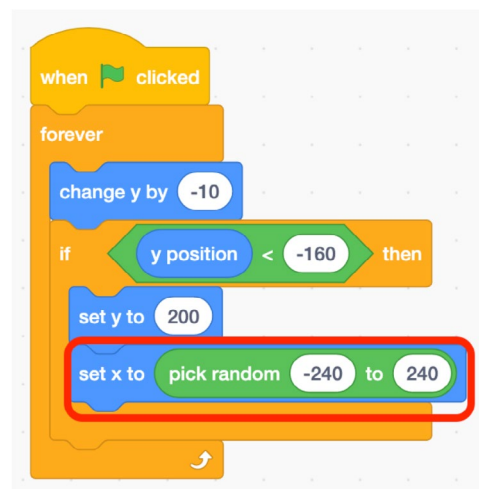
2. **Send it back up:** Now we will add an "if_then_" block to send the sprite back up to the top of the stage when it reaches the bottom. This will involve using the "<" block from operators and the "Y position" block from motion. We can use these blocks to check if the sprite has reached the bottom of the stage. When it does, we will use a "set Y to_" block to send it back to the top of the screen.

Try it out! Your sprite should now fall to the bottom of the screen, reappear at the top of the screen, and continue to fall.



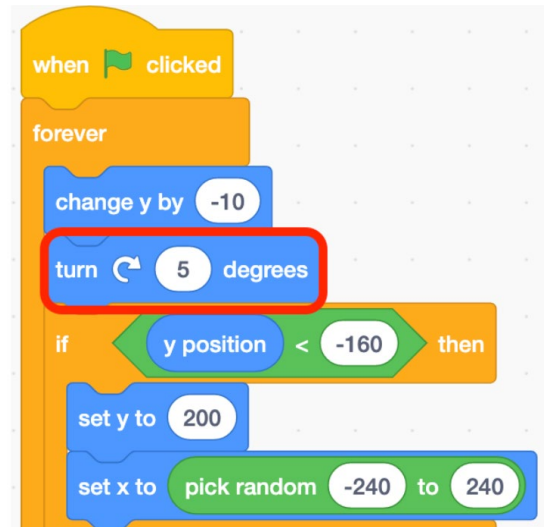
3. **Randomize The Position:** We also want to randomize the **X position** so the sprite doesn't fall in the same place every time. To do this we will use a "pick random _ to _" block from operators and a "set x to_" block. We will add this right after the "set y to_" block.

Now your sprite should fall to the bottom, reappear at the top of the screen at a random **x position** on the top of the screen, and continue to fall.



Project #2: Don't Drop the Donuts (CONTINUED)

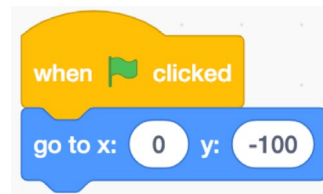
4. **Make it spin:** Just for fun, add a “turn _ degrees” block (either direction) from motion to make your falling item spin. Do this inside your “forever” loop but outside the “if _ then _” statement.



Phase 3: Player Controls

Now we will program movement for the player sprite. In our last project, we programmed movement that was controlled using our keyboard keys. In this project, we will program movement controlled by the mouse.

1. **Set starting position:** Place your player sprite where you want them to start. Then use a “when green flag clicked” and a “go to x_ y_” block to set the starting position for your sprite.



2. **Making movement:** Now we will program movement for our player. This time we will use the mouse to control our layer. We will use a “forever” loop, a “set x to _” block, and the “mouse x” block from sensing.



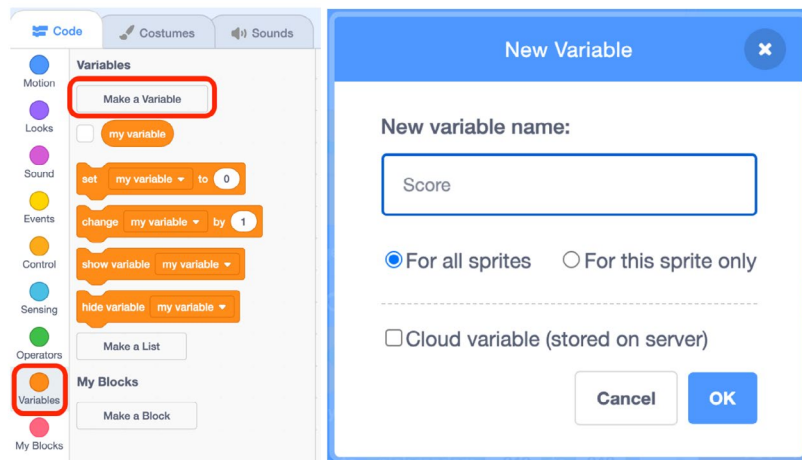
Try it out! Does your sprite follow your mouse? Save your game before moving on!

Project #2: Don't Drop the Donuts (CONTINUED)

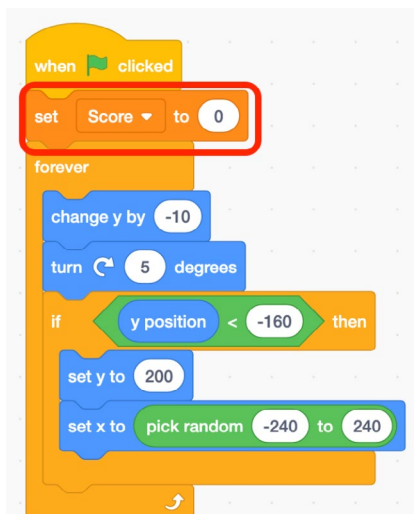
Phase 4: Keeping Score

In order to keep score, we will need to use variables. Variables allow a program to store and keep track of a number.

1. **Make a variable:** Go to the **variables** category in the code block palette and click on **“Make a Variable”**. Call your variable **“Score”** and click **OK**.



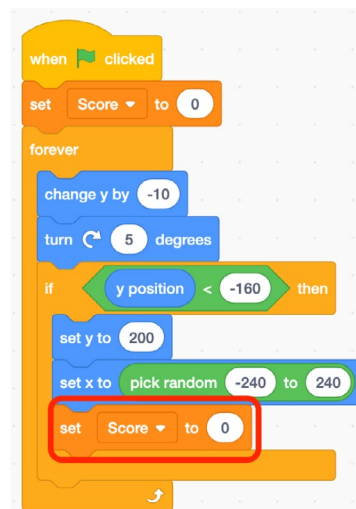
2. **Reset the score:** We need to make sure that when the game starts, our score is set to 0. First, return to the falling object's code. Take a **“set my variable to 0”** block from variables and use the drop down arrow on the block to change **“my variable”** to **“Score”**. Add this block to the top of the falling object's code, directly under the **“when green flag clicked”** block.



3. **Set the rules:** Now it's time to program the rules for this game. Here are the two main rules:

- The player will earn points by catching falling objects
- The player's score will go back to 0 when an object is missed

To make the score reset to 0, we can simply add another **“set score to 0”** block inside the existing **“if_then_”** statement.



Project #2: Don't Drop the Donuts (CONTINUED)

4. **Scoring points:** To let the player catch falling objects to score points, we need to add another “if_then_” block with a “touching_” block from **sensing**. When the player catches a falling object, we need to send the object back to the top of the screen (the same way we do when it hits the ground), and add 1 to the score, using a “change score by 1” block from **variables**.

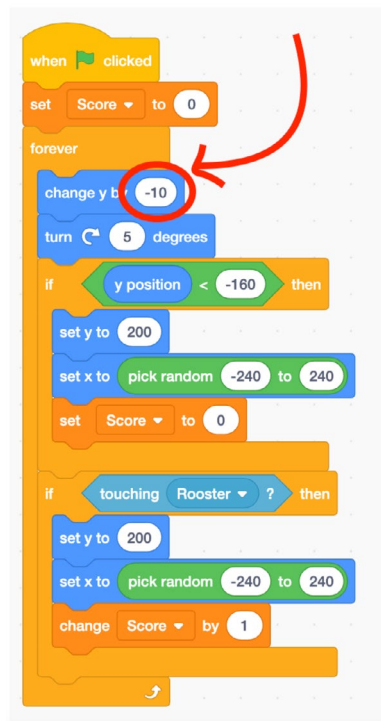
Try it out! Test out your game. Watch the score as you catch (and miss) falling objects. Does everything work the way you want it to? Save your game before moving on!



Phase 5: Finishing Touches

1. **Speed it up:** Right now, this game is probably very easy. We are going to program the game to get harder as the player scores more points. To make the game harder, we will make the objects fall faster.

The **-10** in the “change y by **-10**” block determines how fast our objects fall.

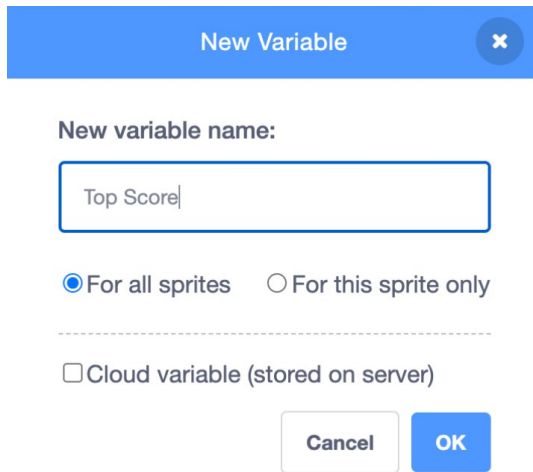


We will need to do some math to make our speed change according to our score. To do this, we will use a **subtraction** block from **operators** and the “Score” block from **variables**.

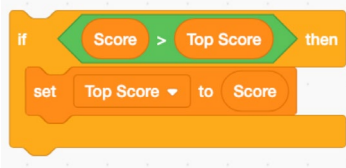


Project #2: Don't Drop the Donuts (CONTINUED)

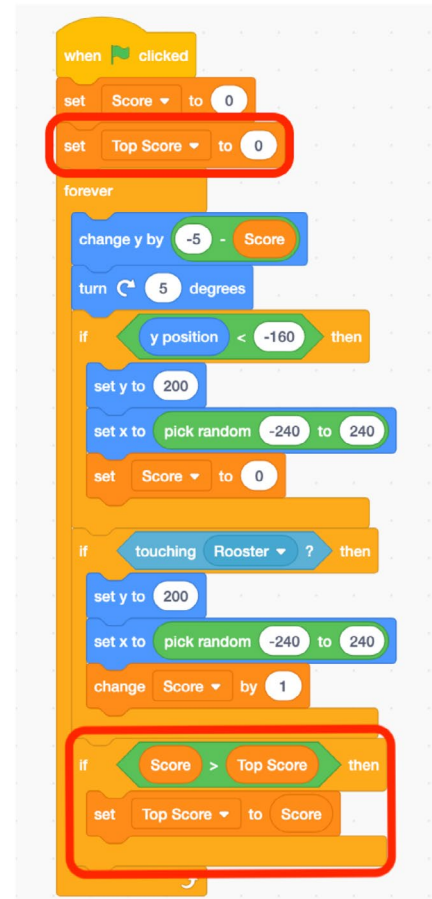
2. **Add a top score:** Lastly, we are going to add a way to keep track of the top score. Go to variables and make another **new variable**, this one called **“Top Score”**.



Next, write the following code to update your top score every time the current score exceeds it.



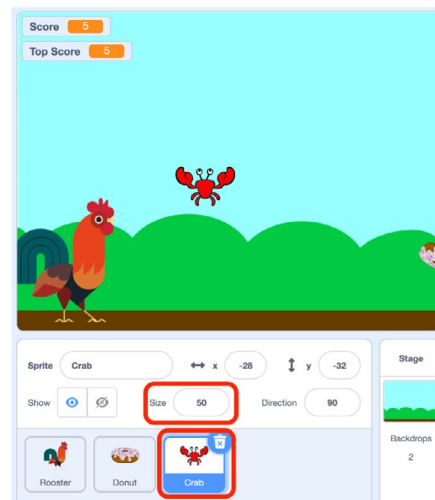
Then, add this code to the bottom of the falling object's **“forever”** loop. Also, reset **Top Score** to 0 at the start of our game.



Phase 6: Bonus Challenge

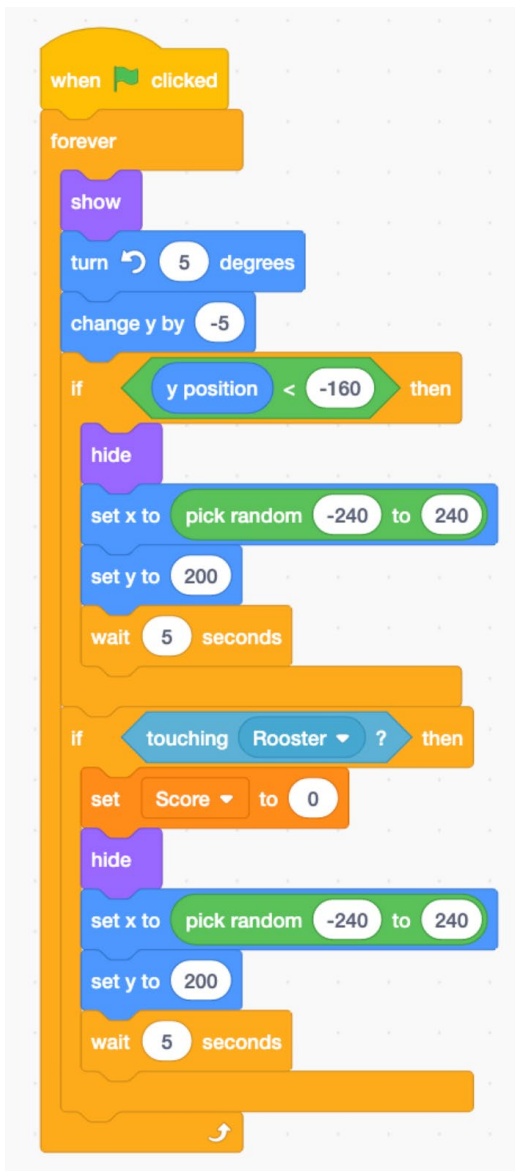
For a bonus challenge, try adding a second falling object that the player should avoid to your game.

1. **Add a new sprite:** Add a new sprite to your project. I will use the **crab sprite** for this. Set the size of this sprite to **50**.



Project #2: Don't Drop the Donuts (CONTINUED)

2. **Code it:** Now will program this sprite. This code be look similar to our falling object code, but with a few differences. This sprite will “wait 5 seconds” after each time it falls, and will use “hide” and “show” blocks to make it invisible while waiting. This sprite will also reset our score to 0 if it is caught by our player.



Playtest

Now that your project is in a playable state, spend some time playtesting it and letting others play it. Here are some questions to guide you while you playtest:

- Does the game work the way you want it to?
- Are there any unexpected behaviors in the game?
- Is the game fun to play?
- Does the game feel too easy, too hard, or just right?
- What could be added to make the game more challenging?
- What could be added to make the game more fun?
- Is there anything confusing or unnecessary that should be removed?

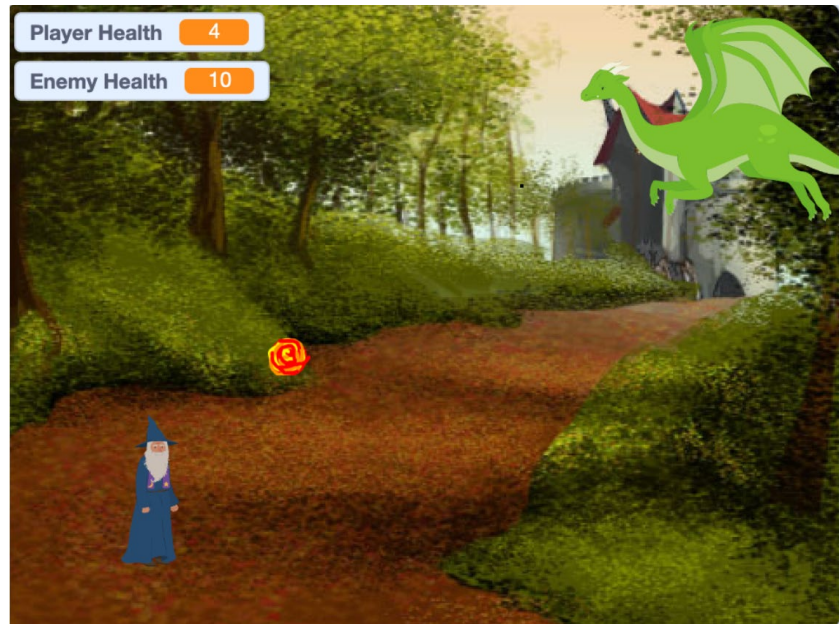
Customize and Iterate

Take your feedback from playtesting and think of ways to improve and customize your project! Here are some ideas on how to take this project further:

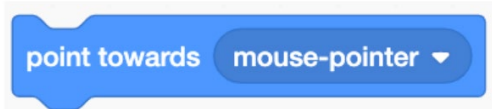
- Change which sprites you are using
- Add sounds to your game
- Add more types of falling objects
- Change the movement controls
- Add a time limit to the game
- Use backdrop changes to show level changes or changes in difficulty
- Improve the detail or design of the backdrops

Project #3: Boss Battle

For our third project, we will **create a version of a video game boss battle**. My battle will be one between a wizard, controlled by the player, and an enemy fire-breathing dragon. You can see (and remix) my version [here](#).

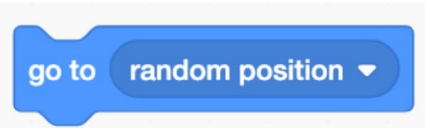


New Blocks Used



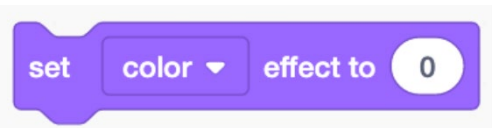
From motion

Rotates the sprite to point towards the mouse pointer or a sprite



From motion

Places the sprite at a random position, another sprite's position, or the mouse pointer

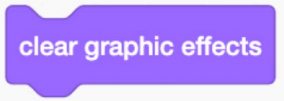
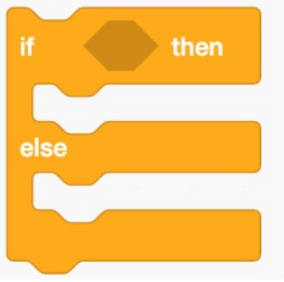


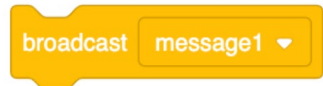
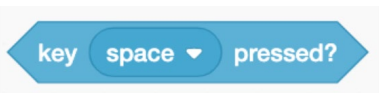
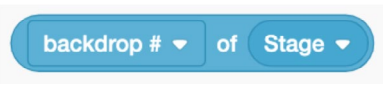


From looks

Sets a graphic effect for a sprite. Click the drop down arrow to see different available effects

Project #3: Boss Battle (CONTINUED)

New Blocks Used (CONTINUED)

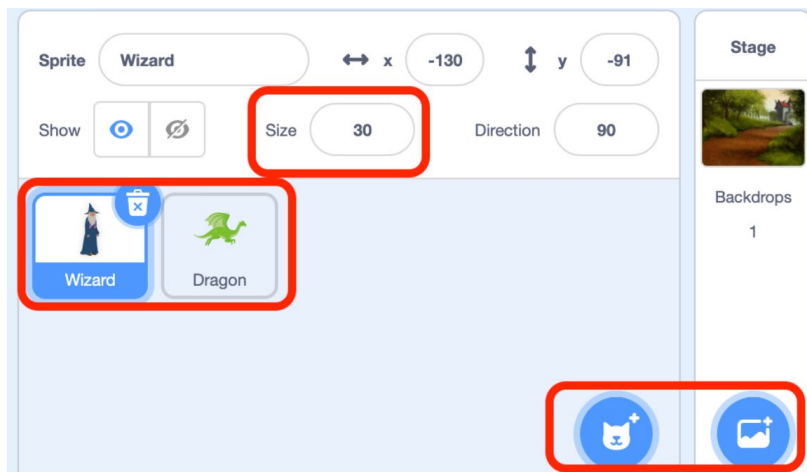
	From looks Clears all graphic effects
	From control Triggers one of two outcomes depending on if the given condition is met
	From control Repeats code inside loop until a condition is met
	From events Receives broadcasted messages from other sprites and triggers attached blocks
	From events Broadcasts a message to be received by other sprites
	From sensing Checks if specified key is pressed
	From sensing Allows sprite to access many types of information about other sprites or stage

Project #3: Boss Battle (CONTINUED)

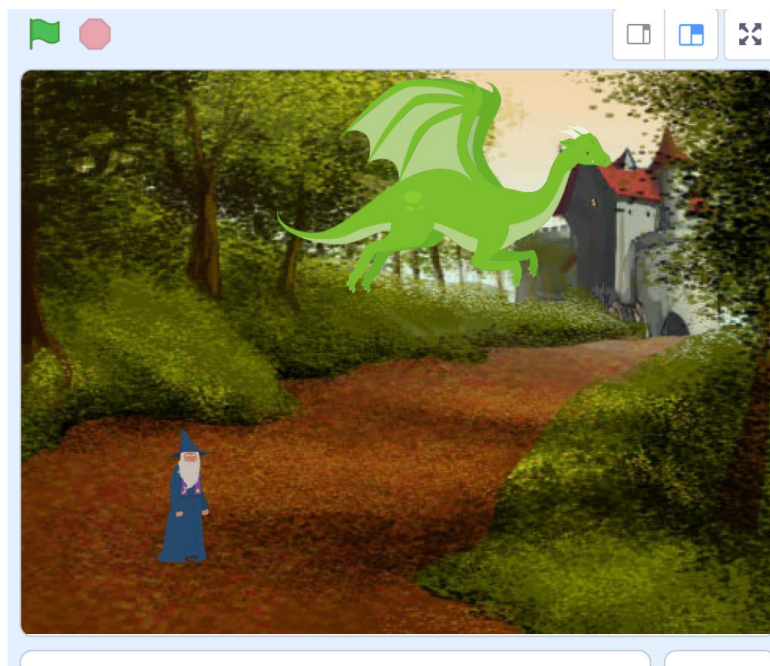
Phase 1: Project Setup

In this first phase, we will open a new project, and pick our backdrop and sprites.

1. **Add your sprites:** Open a new Scratch project by going to scratch.mit.edu, signing into your account, and clicking “**Create**” in the top left corner. Add a player sprite and an enemy sprite to your project. I am using the **wizard** for my player, and the dragon for my enemy. Adjust your sprites to the size you want.



2. **Add a backdrop:** Add a backdrop to your project.



Project #3: Boss Battle (CONTINUED)

Phase 2: Player Movement

Now we will program movement for our player. We will learn a new and improved way to control a player using the keyboard keys.

1. **If/then Movement:** In our first project, we used “when key _ pressed” event blocks to trigger our movement blocks. This works, but creates movement that is not very responsive or smooth. In this project, we will use “if _ then_” blocks with the “key _ pressed?” block from sensing to create smoother movement. We will use the same “point in direction _” blocks and “move 10 steps” blocks that we’ve used for our movement in the past.

```
if key right arrow pressed? then
  point in direction 90
  move 10 steps
```

2. **Loop it:** For this code to work properly, we need to put it inside a “forever” loop to constantly check if the key is being pressed. We will start with a “when green flag clicked.”

```
when green flag clicked
  forever
    if key right arrow pressed? then
      point in direction 90
      move 10 steps
```

3. **Two directions:** Add another “if _ then_” loop to program movement to the left.

```
when green flag clicked
  forever
    if key right arrow pressed? then
      point in direction 90
      move 10 steps
    if key left arrow pressed? then
      point in direction -90
      move 10 steps
```

4. **Two directions:** Add another “if _ then_” loop to program movement to the left.

```
when green flag clicked
  go to x: -150 y: -100
  set rotation style left-right
  forever
    if key right arrow pressed? then
      point in direction 90
      move 10 steps
    if key left arrow pressed? then
      point in direction -90
      move 10 steps
```

→ **Test your code** by clicking on the green flag and using your arrow keys to move!
Make sure your project is saved.

Project #3: Boss Battle (CONTINUED)

Phase 3: Enemy Movement

Now we will program movement for our enemy sprite.

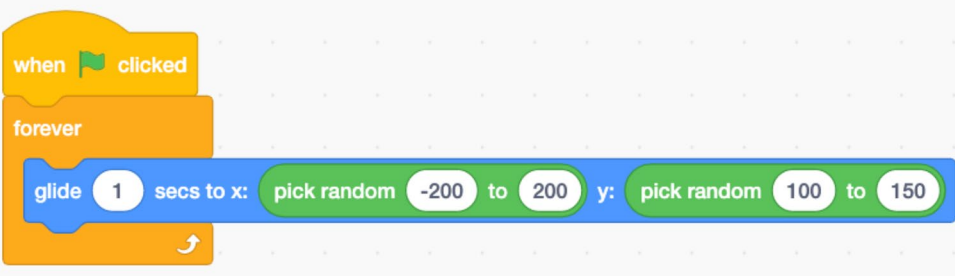
1. **The Power of Randomness:** In video games, **randomness** is extremely important for programming unpredictable events and behaviors, which are important for making games challenging and fun. We will use the “**pick random _ to _**” block along with a “**glide 1 secs to x_ y_**” to program movement for our enemy sprite.



```
glide 1 secs to x: pick random -200 to 200 y: pick random 100 to 150
```

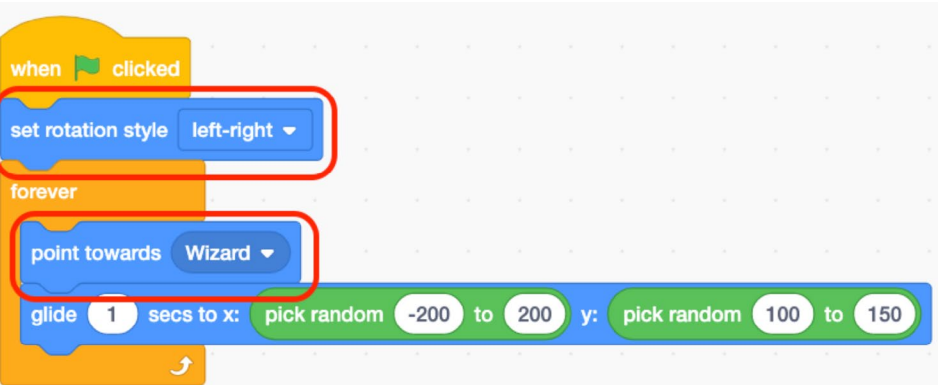
This code will send our sprite to a random X,Y position near the top of the stage every second.

2. **Loop it:** We will place this block in a “**forever**” loop to repeat it. We will again start with a “**when green flag clicked**”



```
when green flag clicked  
forever  
  glide 1 secs to x: pick random -200 to 200 y: pick random 100 to 150
```

3. **Turn and face:** Add a “**point towards _**” block to make the enemy face our player.



```
when green flag clicked  
set rotation style left-right  
forever  
  point towards Wizard  
  glide 1 secs to x: pick random -200 to 200 y: pick random 100 to 150
```

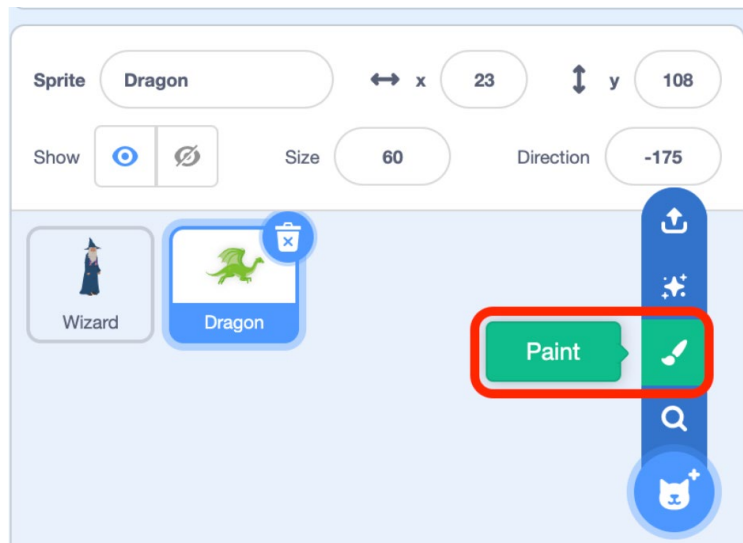
→ **Try it out!** Click the **green flag** and watch your enemy start to fly around!

Project #3: Boss Battle (CONTINUED)

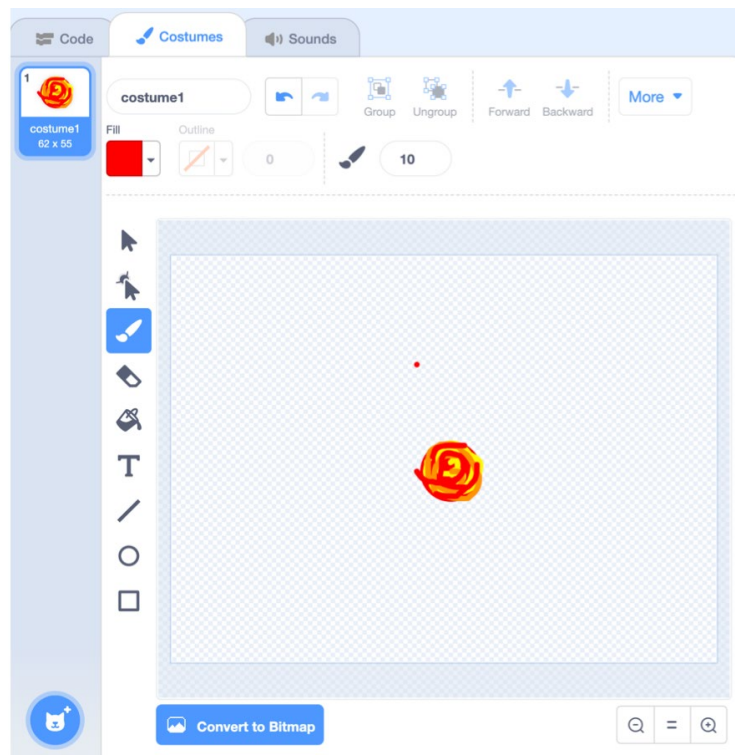
Phase 4: Under Attack!

Now we will create a fire-ball attack for our enemy sprite!

1. **Paint a new sprite:** Hover over the **Choose a Sprite** button and click on the paint brush to open the paint menu.



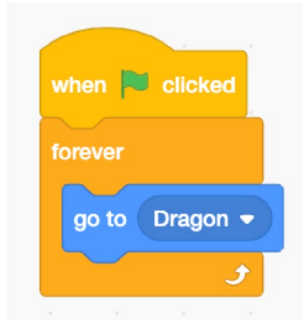
2. **Paint your sprite:** Use the paint tool to paint a fireball sprite! Shrink your sprite using the size field in the sprite menu if necessary.



Project #3: Boss Battle (CONTINUED)

3. **Program the Fireball:** The fireball needs to start at the enemy sprite. Let's start with a "when green flag clicked", a "forever" loop and a "go to _" block to do this. Select your enemy sprite in the "go to" block.

Press the **green flag** to try it out. This will make the fireball appear at the center of the enemy sprite. We'll fix it and make it appear at the front of our sprite in the next step.

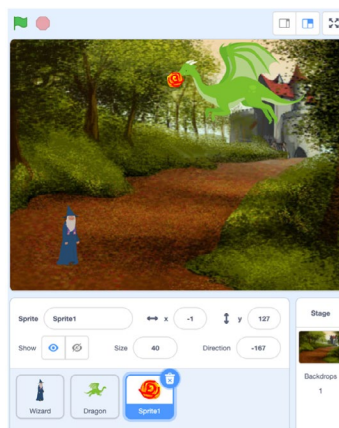


4. **Adjust the position:** We need to move our fireball to the front of our enemy sprite. Where we place it will be determined by which way our enemy sprite is facing, which we can determine using the following block from **sensing**:

This extremely useful block can be used to get information about other sprites or the stage. Select your enemy sprite using the second dropdown, and then "direction" using the first.



Now we will need to use a ">" block inside an "if _ then _ else _" block to position our fireball depending on which direction the enemy sprite is facing. Use "change x by _" and "change y by _" blocks to place the fireball sprite where you want to. Note the use of positive and negative values in the "change x by _" blocks.



→ Now the fireball stays at the front of the enemy sprite where-ever it goes.

Project #3: Boss Battle (CONTINUED)

5. **Aim, fire!:** Now we will program the fireball to aim and fly at our player sprite. We will use a “**point towards _**” block, and a “**move 10 steps**” block inside of a loop. We will use a “**repeat until_**” loop to keep the fireball moving until it’s **Y position** is too low.

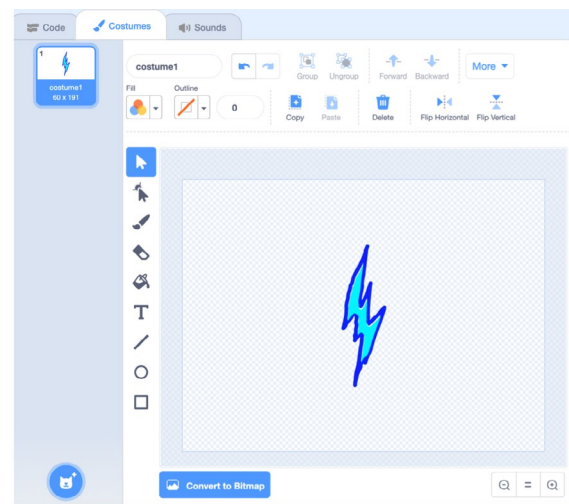
Try it out! Press the **green flag** and watch fireballs rain down from our enemy sprite onto our player sprite.



Phase 5: Fight Back!

Now we are going to create a lighting-bolt attack for our player sprite.

1. **Paint a new sprite:** Hover over the **Choose a Sprite** button and click on the paint brush to open the paint menu. Paint a new sprite to be your player’s attack.



Project #3: Boss Battle (CONTINUED)

2. **Make it fly:** We will start with code that is similar to what we just wrote for the lightning fireball. The lightning bolt will start at the player sprite, and fly upward until it nears the top of the stage.

```
when green flag clicked
  forever loop
    go to Wizard
    repeat until y position > 150
      change y by 10
```

3. Next we'll add a "wait until_" block and some "hide"/"show" blocks to make the lightning bolts only shoot and show when the spacebar is pressed.

```
when green flag clicked
  forever loop
    hide
    wait until key space pressed?
    go to Wizard
    show
    repeat until y position > 150
      change y by 10
```

Phase 6: Health

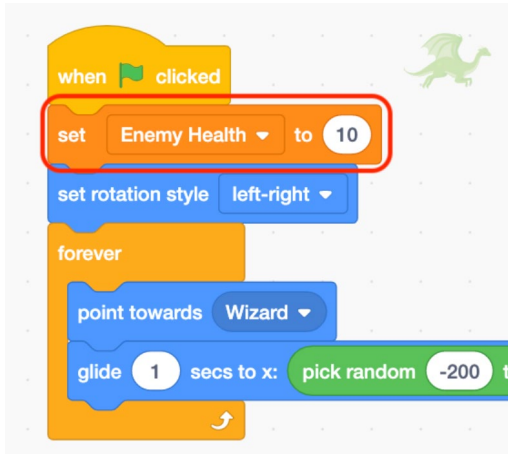
Now we will create variables for player health and enemy health, and program these to interact with our attacks.

1. **Create variables:** Go to **variables** and create two new variables: "Enemy Health" and "Player Health".

The screenshot shows the Scratch Variables palette. The 'Variables' category is selected. The 'Make a Variable' button is highlighted with a red box. Below it, two variables are listed: 'Enemy Health' and 'Player Health', both with checkboxes checked and highlighted with red boxes. Below the variables, there are blocks for 'set Enemy Health to 0', 'change Enemy Health by 1', 'show variable Enemy Health', and 'hide variable Enemy Health'. At the bottom, there are buttons for 'Make a List' and 'My Blocks' with a 'Make a Block' button.

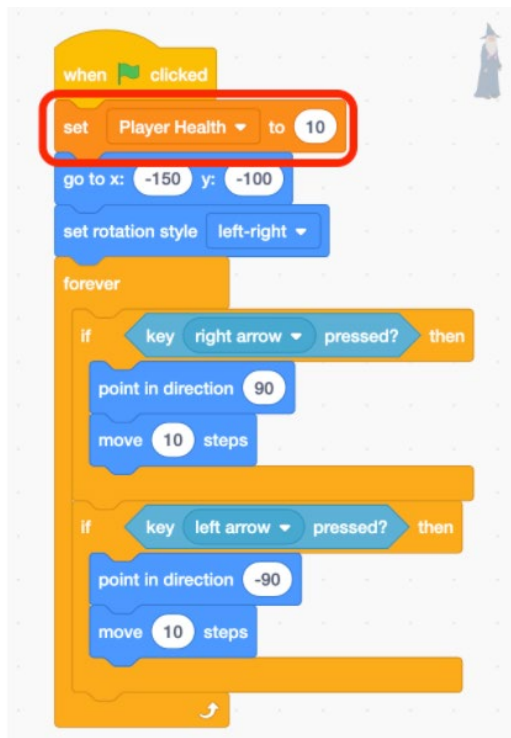
Project #3: Boss Battle (CONTINUED)

2. Set starting health: Go to the player sprite and add a “set Player Health to 10” to the start of the code, underneath the “when green flag clicked”.



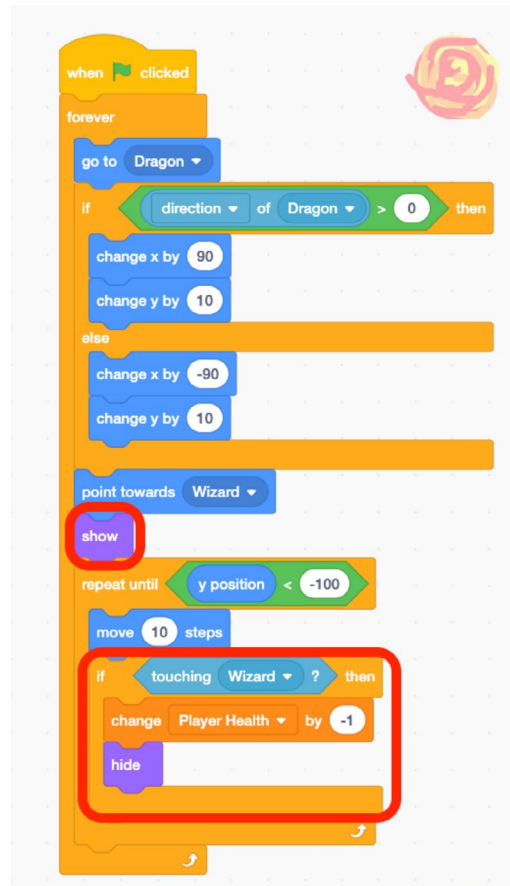
The image shows a Scratch code editor for a dragon sprite. The code starts with a yellow 'when green flag clicked' block. Below it is an orange 'set Enemy Health to 10' block, which is highlighted with a red box. This is followed by a blue 'set rotation style to left-right' block. A yellow 'forever' loop contains a blue 'point towards Wizard' block and a blue 'glide 1 secs to x: pick random -200 to 200' block.

3. **Enemy Health:** Do the same for the enemy sprite, this time setting “Enemy Health” to 10.



The image shows a Scratch code editor for a wizard sprite. The code starts with a yellow 'when green flag clicked' block. Below it is an orange 'set Player Health to 10' block, highlighted with a red box. This is followed by a blue 'go to x: -150 y: -100' block, a blue 'set rotation style to left-right' block, and a yellow 'forever' loop. Inside the loop, there are two conditional blocks: 'if key right arrow pressed? then' followed by 'point in direction 90' and 'move 10 steps'; and 'if key left arrow pressed? then' followed by 'point in direction -90' and 'move 10 steps'.

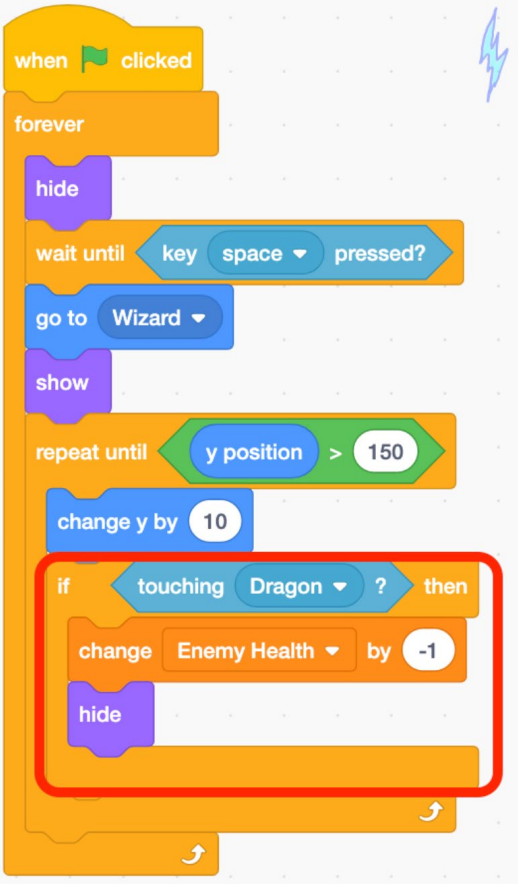
4. **Program Player Health:** Go to the fireball sprite’s code. Add an “if_then_” block to decrease the player’s health and then “hide” the fireball if the fireball touches the player. Make sure you add this inside the “repeat until” loop. Also add a “show” block to make the fireball reappears when it starts moving.



The image shows a Scratch code editor for a fireball sprite. The code starts with a yellow 'when green flag clicked' block. Below it is a yellow 'forever' loop. Inside the loop, there is a blue 'go to Dragon' block, followed by an 'if direction of Dragon > 0 then' conditional block. The 'then' branch contains 'change x by 90' and 'change y by 10'. The 'else' branch contains 'change x by -90' and 'change y by 10'. Below these is a blue 'point towards Wizard' block, a purple 'show' block (highlighted with a red box), and a yellow 'repeat until y position < -100' loop. Inside this loop is a blue 'move 10 steps' block, followed by an 'if touching Wizard? then' conditional block. The 'then' branch contains 'change Player Health by -1' and a purple 'hide' block (both highlighted with a red box).

Project #3: Boss Battle (CONTINUED)

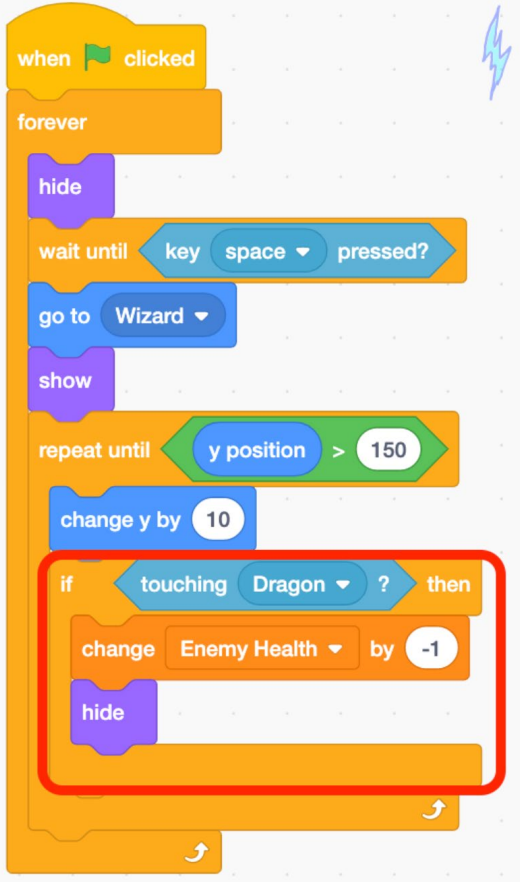
5. **Enemy Health:** Do the same for the enemy sprite, this time setting “Enemy Health” to 10.



The code for step 5 is as follows:

```
when green flag clicked
  forever loop
    hide
    wait until key space pressed?
    go to Wizard
    show
    repeat until y position > 150
      change y by 10
      if touching Dragon ? then
        change Enemy Health by -1
        hide
```

6. **Program Enemy Health:** Go to the **lightning bolt sprite's code**. Add an “if_then_” block to decrease the enemy’s health and then “hide” the lightning if the bolt touches the enemy. Make sure you add this inside the “repeat until” loop.



The code for step 6 is as follows:

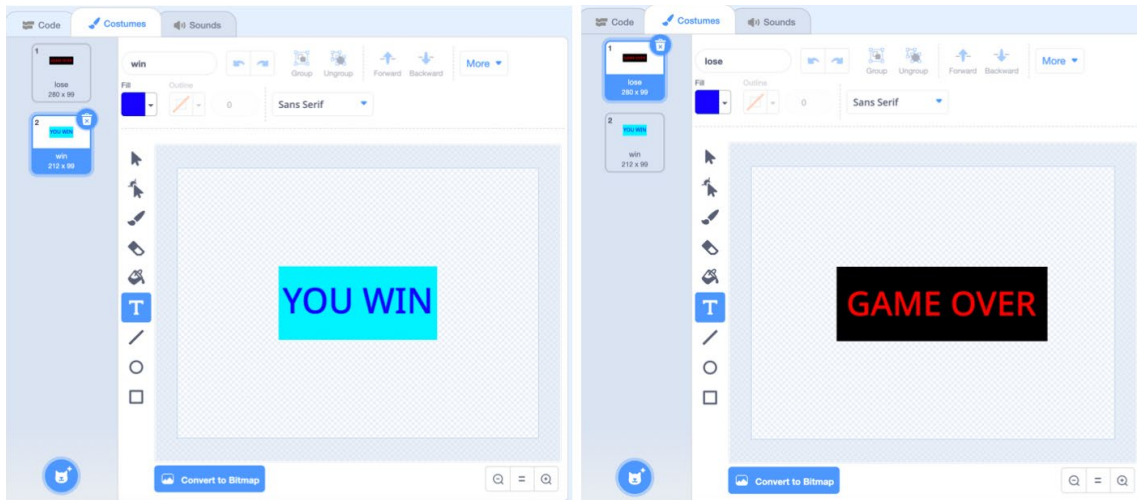
```
when green flag clicked
  forever loop
    hide
    wait until key space pressed?
    go to Wizard
    show
    repeat until y position > 150
      change y by 10
      if touching Dragon ? then
        change Enemy Health by -1
        hide
```

Project #3: Boss Battle (CONTINUED)

Phase 7: Finishing Touches

Now we will program the end of our boss battle, which will occur when either sprite runs out of health.

1. **Create banner sprite:** We are going to create a new sprite to display either “you win” or “game over” when the game ends. To do this, paint a new sprite, and create two costumes for it- one for each outcome. Title the costumes “win” and “lose”.



2. **Reaching the end:** Write the following code to switch to the appropriate costume and display the sprite when either the player or enemy are out of health. →



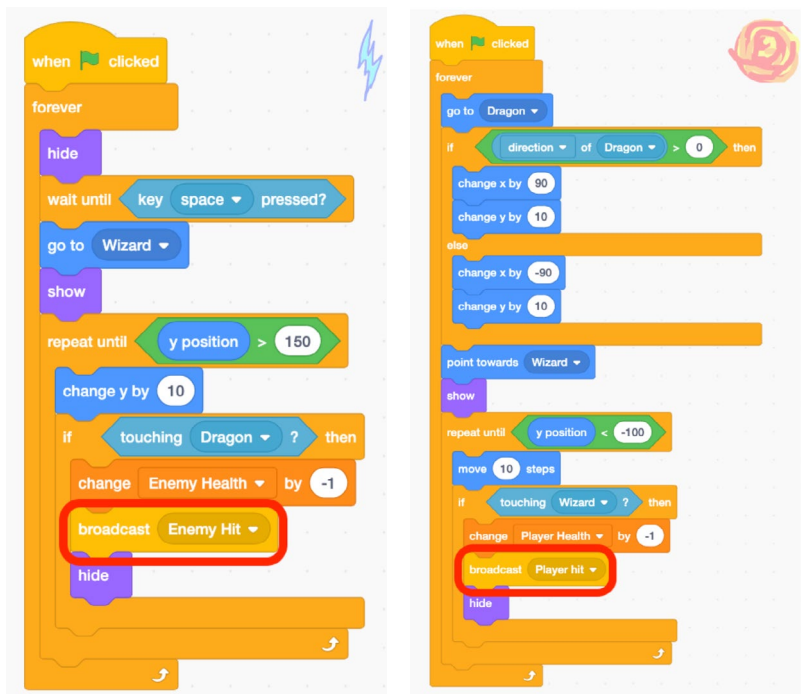
3. **Try it out!** Now that all of the pieces of your game are in place, try it out and see if you can defeat the enemy!

Project #3: Boss Battle (CONTINUED)

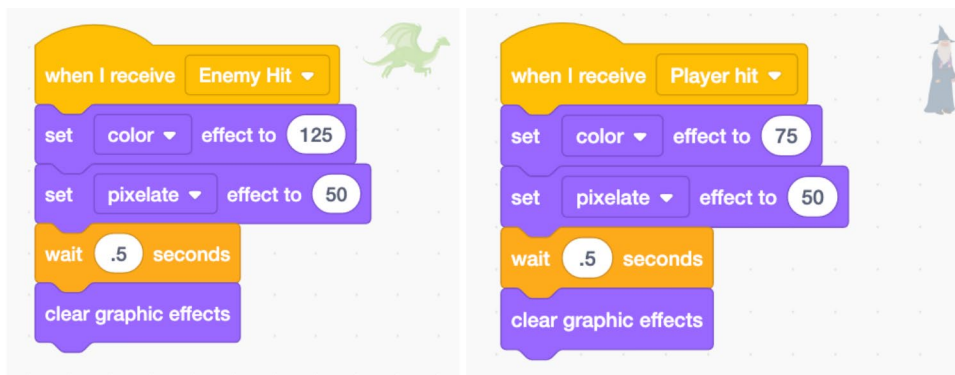
Phase 8: Bonus Challenge:

For a bonus challenge, create some animations for our player and enemy sprite when they get hit by an attack.

1. **Broadcast:** “Broadcast” and “when I receive_” blocks from events allow us an easy way to trigger events across sprites. Send a message with a “broadcast” block in one sprite, and trigger other blocks with a “when I receive” block in another sprite. Add a “broadcast” block to the lightning bolt script just after the “change Enemy Health by -1” block. Broadcast a new message called “Enemy hit” by clicking on the dropdown arrow in the block, clicking “new message”, and naming the message. Then do the same in the fireball sprite with a message called “Player hit”



2. **Receive the broadcast:** In both the player and enemy sprites, use a “when i receive _” to receive the proper message and apply **graphic effects**, costume changes, or anything else! Use a “wait _secs” block and a “clear graphic effects” block from looks set your sprite back to normal at the end of the animation.



Project #3: Boss Battle (CONTINUED)

Playtest

Now that your project is in a playable state, spend some time playtesting it and letting others play it. Here are some questions to guide you while you playtest:

- Does the game work the way you want it to?
- Are there any unexpected behaviors in the game?
- Is the game fun to play?
- Does the game feel too easy, too hard, or just right?
- What could be added to make the game more challenging?
- What could be added to make the game more fun?
- Is there anything confusing or unnecessary that should be removed?

Customize and Iterate

Take your feedback from playtesting and think of ways to improve and customize your project! Here are some ideas on how to take this project further:

Change which sprites you are using:

- Add sounds to your game
- Add more types of attacks the player/enemy
- Change the movement controls for the player
- Change the movement program for the enemy
- Add levels that precede this boss battle
- Improve the detail or design of the backdrops



Final Project: Build your own Impact Game!

Now that you've built three games in Scratch (and from scratch!), it's time to use your new skills to build a game for impact.

What is an Impact Game?

Impact games are games that deal with real issues and encourage players to learn and take action on important issues in the real world! These can be issues big and small.

Steps:

1. **Choose your issue:** What is the real world issue or challenge that you will tackle with your impact game? It can be something relating to your local community, or something that affects the whole world.
2. **Dive into your issue:** Who is impacted by your issue? What causes it to occur? Where and when does it happen? What are possible solutions to the issue? What information should the public know about this issue and its possible solutions?
3. **Brainstorm your game:** Can you think of a connection between your chosen issue and any games you've played or made? Decide what type of impact game you're going to make. Your game can be built off of one of the games we've already made, something you create from scratch, or a remix of an existing project.
4. **Plan your game:** What sprites are you going to need? What will each sprite do? What backdrops will you need? What will the player's experience be like? Make a plan for making your project before you start programming.
5. **Make your game:** Get into Scratch and build your game!
6. **Playtesting:** Let someone else play your game and get them to give you feedback. Look for bugs or unexpected behaviors in the game.
7. **Improve:** Put your playtesting feedback into action by fixing bugs and improving your game.

