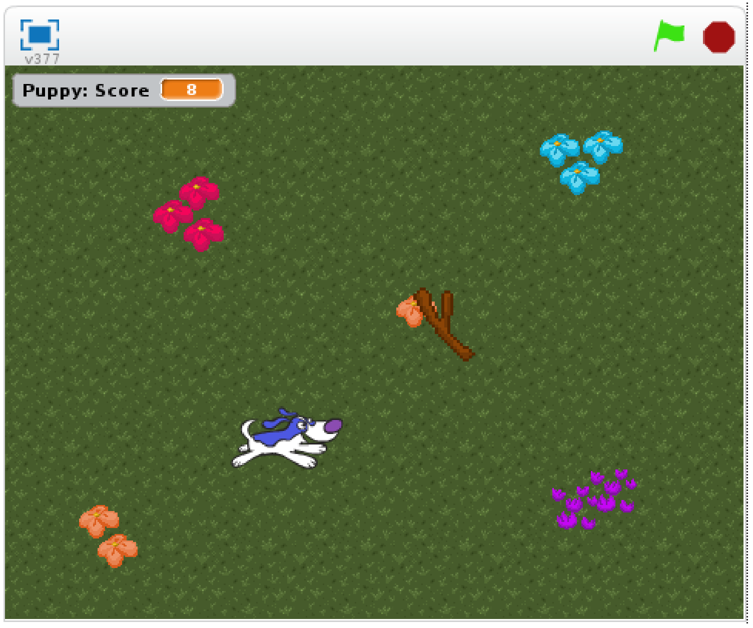
Let's Make Pickin' Sticks

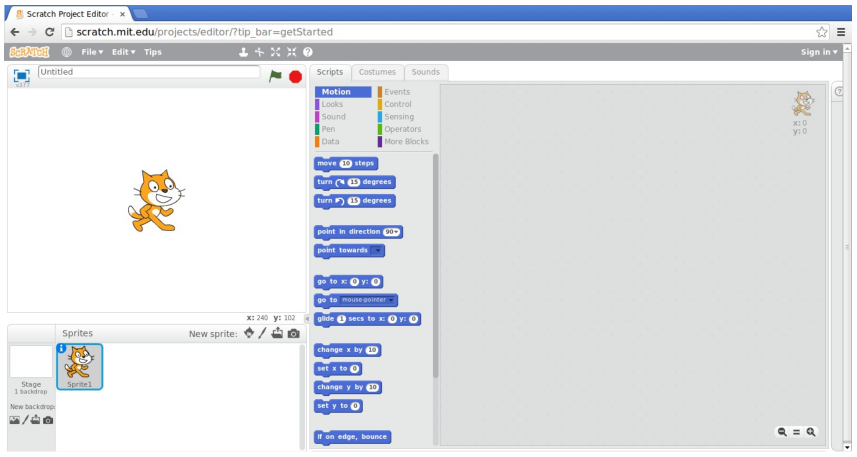


Pickin' Sticks is a simple game you can make to become familiar with:

- Having multiple sprite objects
- Collision Detection
- The X/Y Coordinate Plane
- Random number generator
- Moving around with the keyboard
- Variables and score
- Control statements

1. Create!

Click on the “Create” link to start a new project. You will have a blank screen with Scratch the Cat.



Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

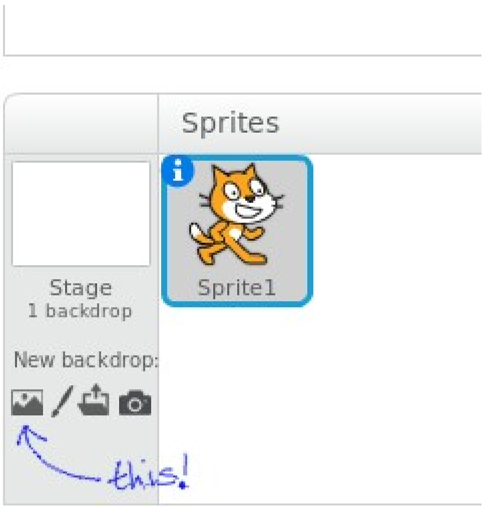
2. Choose a Backdrop!

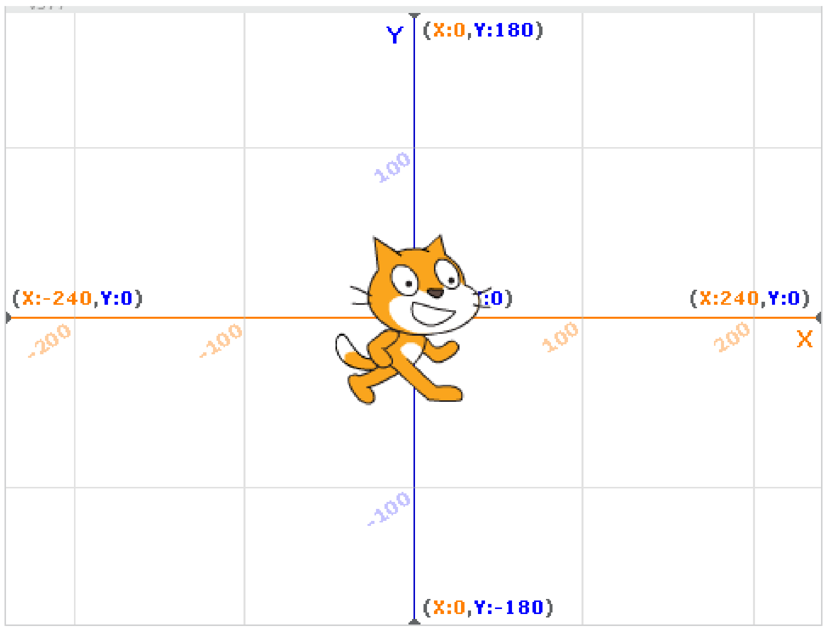
On the lower-left of the screen, select the picture icon under "New backdrop:"

Scratch has a lot of pre-made backdrops for you to use! You can also draw your own.

We are going to add two pre-made backdrops for now: Choose one that you like, and then go to the **Other** category and select **xy-grid.**

We will use the xy-grid temporarily, because it helps us understand how to position our sprites on the screen.

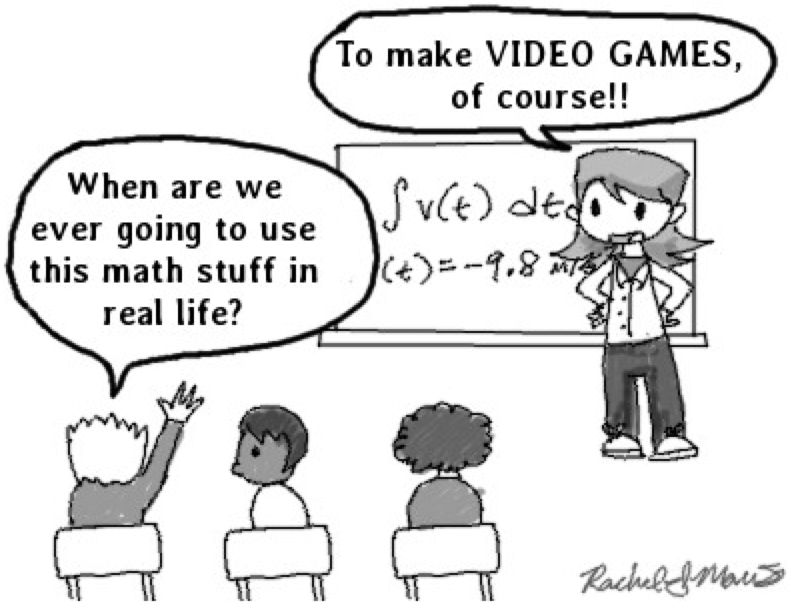




This coordinate plane is similar to what you would see in Algebra class. Notice that the very left side of the screen is at the X coordinate -240. The right side is at positive 240.
The vertical axis goes from -180 to 180.

So, if we wanted to move a character **LEFT**, we would subtract from their X coordinate.
If we wanted to move a character **UP**, we would add to their Y coordinate.
Scratch the Cat is currently at **(0,0)**.

Document created by Rachel J. Morris, November 2013 Rachel@Moosader.com
Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/


When are we ever going to use this math stuff in real life?
To make VIDEO GAMES, of course!!
∫v(t) dt
(t)=-9.8 m/s
Rachel J Morris

Math in Video Games & Computer Science

Math is great, especially when it comes to video games! Here are some of the things you might use math for:

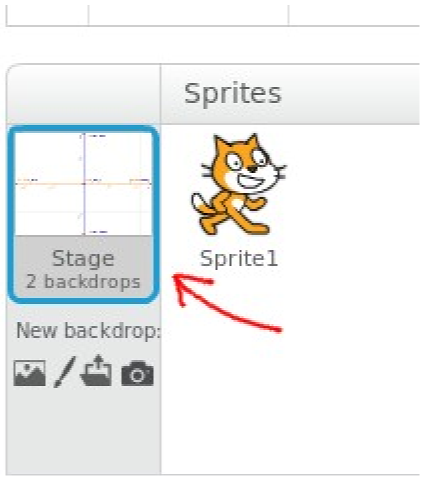
- **Coordinate Planes:** Moving objects around a 2D and 3D coordinate system.
- **Physics & Calculus:** Newton's laws of motion can be written as functions, and you can figure out Acceleration, Velocity, and Position with integrals and derivatives! Hmm... Acceleration, Velocity, and Position might be useful for many types of games!
- **Trigonometry:** Rotating sprites and 3D objects. Handling a first-person camera movement.
- **Algebra:** Programming has variables, just like Algebra! Programming has functions, just like Algebra!
- **The Pythagorean Theorem:** This theorem is useful when we want to find the distance between two objects, and that could be useful for a game!

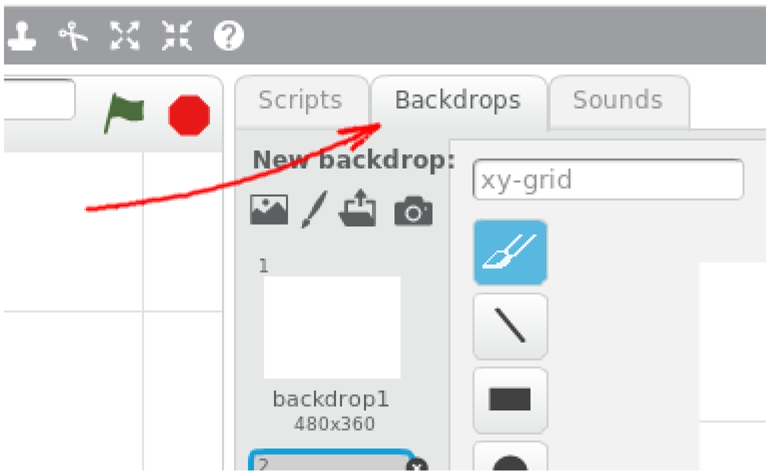
Of course with Scratch, we're only going to worry about coordinate planes (and, if you want to implement it, custom physics!). If you're interested in game *programming* as a profession, or if you want to get a degree in *computer science*, you'll find that these can be math-heavy fields!

But don't worry! You can tackle any level of math, if you build a strong foundation in basic math concepts and continue building your math knowledge through classes and practice!

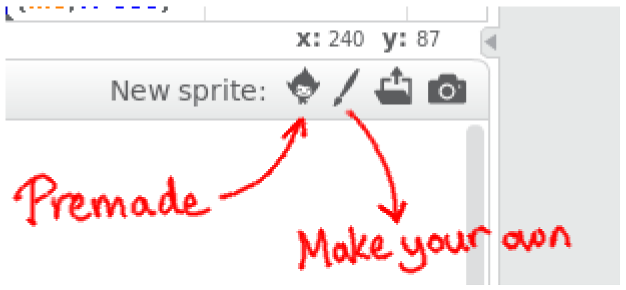
Document created by Rachel J. Morris, November 2013 Rachel@Moosader.com
Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

Make sure that the **xy-grid** is your current backdrop. To change backdrops, click on the backdrop thumbnail in the lower-left, and then on the "Backdrops" tab on the script pane.

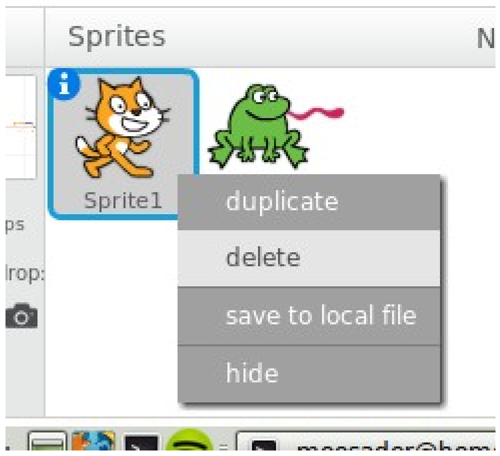




3. Choose a Player and an Item sprite!

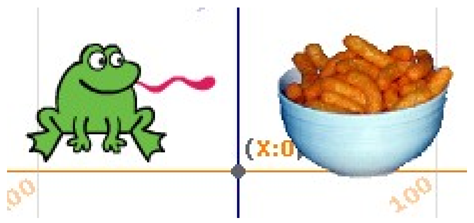


Next, let's choose a sprite. You can use Scratch the Cat if you would like, but you can also click the New Sprite icon to choose a pre-made sprite. When you design your own game, you can draw a new sprite but for now, let's just stick to pre-made.



When you choose a new sprite, you might want to get rid of the cat. Right-click on the cat sprite, then click "delete".

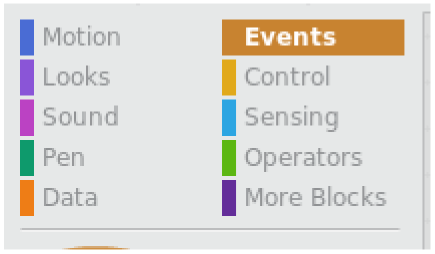
4. Choose a second sprite. In my original "Pickin' Sticks" game, you pick up sticks! But, find a sprite in the pre-made library and choose something for your character to pick up!



Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

Now that we have our player sprite (I have a frog) and our pick-up object (I have cheesy puffs), let's implement some scripts!

5. Move the Player!

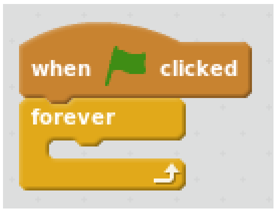


Click on your **Player** sprite, then click the "Scripts" tab on the big square pane. You will see a bunch of categories for different types of functions we can use. First, click on **Events.**

Click and drag the **"When [green flag] clicked"** block into the grey window.

Notice that most of these blocks have a rounded top corner. This indicates that this is a **starting** block. Most blocks have puzzle-piece indentations or bumps, which show that they can snap together. We can add functionality that occurs when the green flag is clicked.

When the green flag is clicked, this is essentially the mark that our game has started. When the game starts, we want it to keep **looping.** This way, we can check for player input over and over.



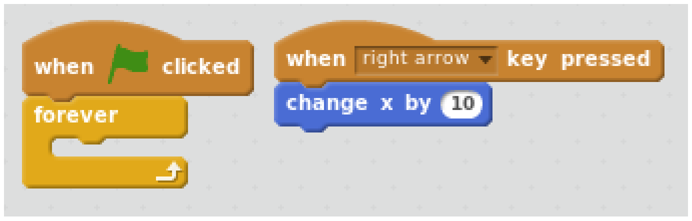
Go to the **Control** blocks, and drag the **forever** block to be joined with our **When [green flag] clicked** block.

Think of this as the main game loop. Anything before our **forever** loop is initialization – setup, getting the game ready to go for the first time.

Within our game loop, we want to check for:

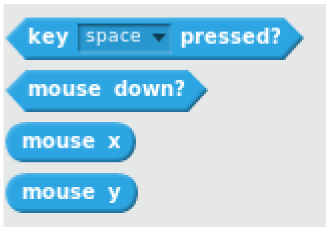
- Is the player pressing a key?
- Has the player gotten to our pick-up item?
  - If they have, add to their score!
  - Move the pick-up item somewhere else

How do we check for keyboard input? Well, there's the **Event > when ___ key pressed** block, but it doesn't feel very smooth when we hold down an arrow key! You can try it out if you would like:



Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

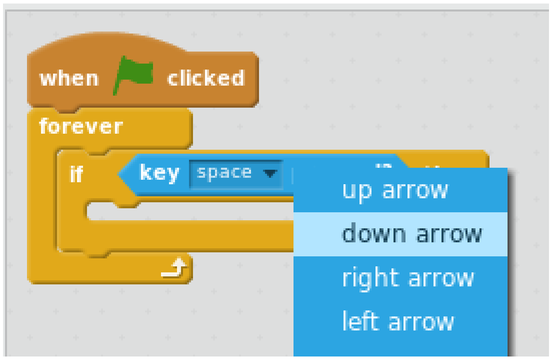
But, we want nice, smooth movement. Instead of using the Event block, we need to use a **Sensing** block instead.



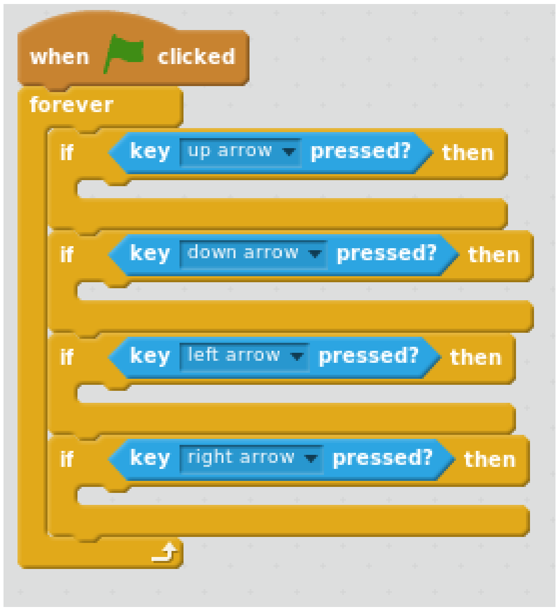
Notice that the sensing blocks are a funny shape: they're pointy on both sides. This means that they need to go **inside** of another block, we can't use these as standalone blocks!

Go to the **Control** blocks, and choose the **if then** statement. Drag it to *inside* of the **forever** loop block.

Then, within the empty diamond slot in our **if then** block, we will insert the **Sensing: key _ pressed?** Block. By clicking on the tiny down-arrow, we can set which key we're looking for.



We need to create four of these, for each direction! Right-click on the **if then** block, and click “duplicate”. Drag each duplicate after the previous **if then** block, and set their keys to **up arrow**, **down arrow**, **right arrow**, and **left arrow**.


when clicked
forever
if key up arrow pressed? then
if key down arrow pressed? then
if key left arrow pressed? then
if key right arrow pressed? then

Now, within the game loop, we are checking for whether any of the directional arrow keys are pressed.

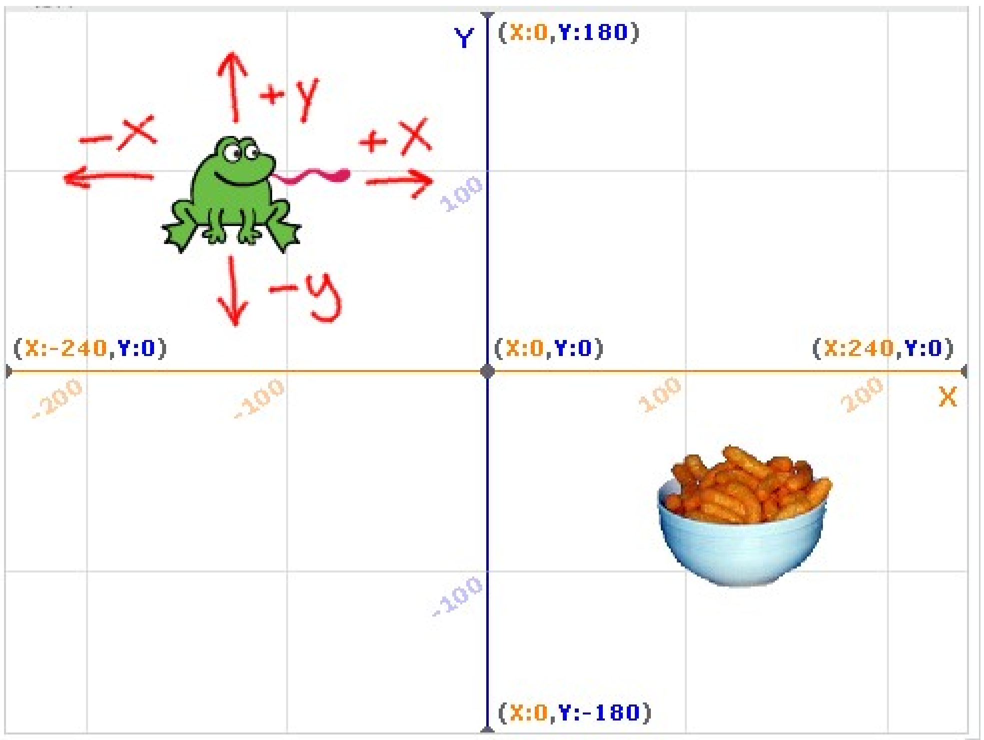
To go UP, we add to the Y coordinate.

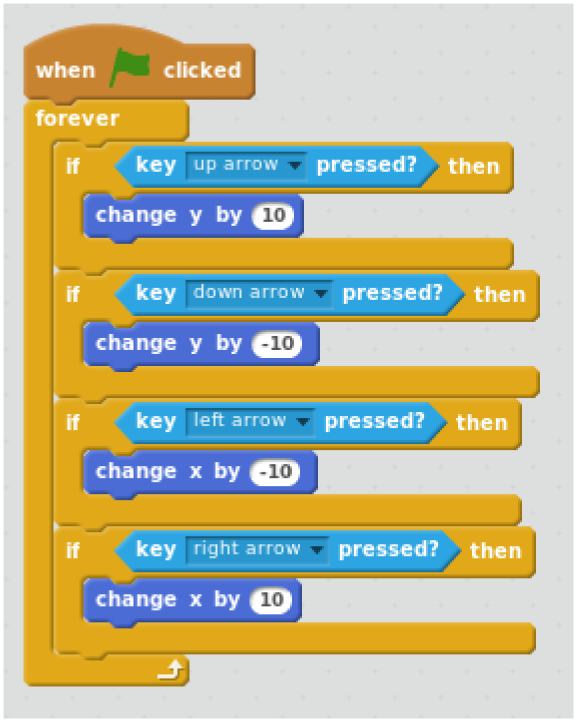
To go DOWN, we subtract from the Y coordinate.

To go LEFT, we subtract from the X coordinate.

To go RIGHT, we add to the X coordinate.

Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/



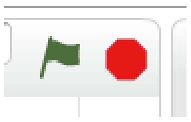


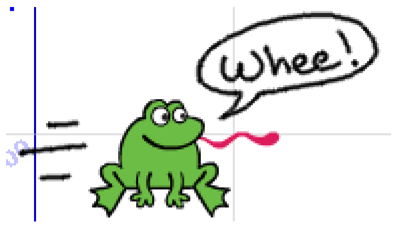
```
when [flag] clicked
forever
    if <key (up arrow) pressed?> then
        change y by (10)
    if <key (down arrow) pressed?> then
        change y by (-10)
    if <key (left arrow) pressed?> then
        change x by (-10)
    if <key (right arrow) pressed?> then
        change x by (10)
```

Under **Motion**, select the **change y by 10** block. Use this block in the **up** and **down** key if statements. Change the value to -10 for the **down** direction.

Then, use the **change x by 10** block for **left** and **right**. Set the amount to -10 for the **left** direction.

Now, click on the green flag to test out the game. Hold down the arrow keys and see how smooth your player moves!





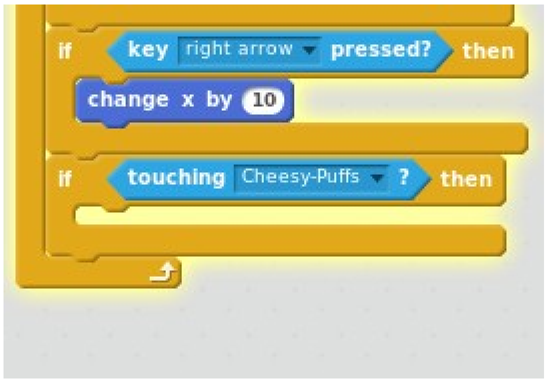
Document created by Rachel J. Morris, November 2013 Rachel@Moosader.com
Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

6. Pick up the Item!

Now, how can we tell if the player is touching the item? This is a difficult concept to explain in C++, but we're using Scratch so it will be easy!

Go to the **Control** blocks and create a new **if then** statement, after our keyboard **if then** statements.

Under **Sensing**, there is a handy little block for seeing if two objects are touching. Drag the top-most Sensing block into the if _ then slot, and use the drop-down menu to select the name of your item sprite.

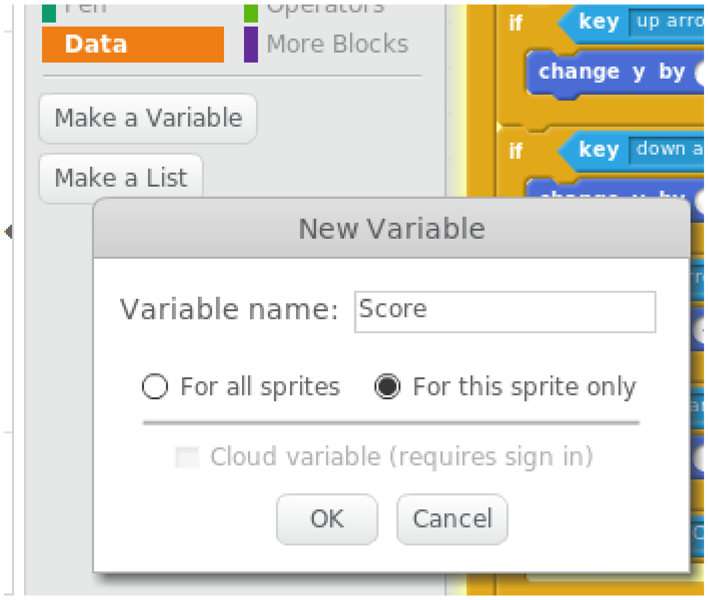

if key right arrow pressed? then
change x by 10
if touching Cheesy-Puffs ? then

Once we pick up the item, we'll want to add to the player's score.

But wait – we don't have a score!

Don't panic, it will be alright. I'll give you a second, just calm down. WE CAN DO IT, STOP FLAILING!

Click on the **Data** category. There are two buttons here: Make a Variable, and Make a List. A list might be useful if we were picking up multiple items, and wanted to keep track of what they were. But, we just want a simple score counter, so let's make a variable.



New Variable

Variable name: Score

For all sprites [✓ For this sprite only]

Cloud variable (requires sign in)

OK Cancel

Click on **Make a Variable** and name it “Score”. Look at the two little radio buttons, make sure to select “For this sprite only”. (We are currently working within the Player sprite, and the Items won't have a score!)

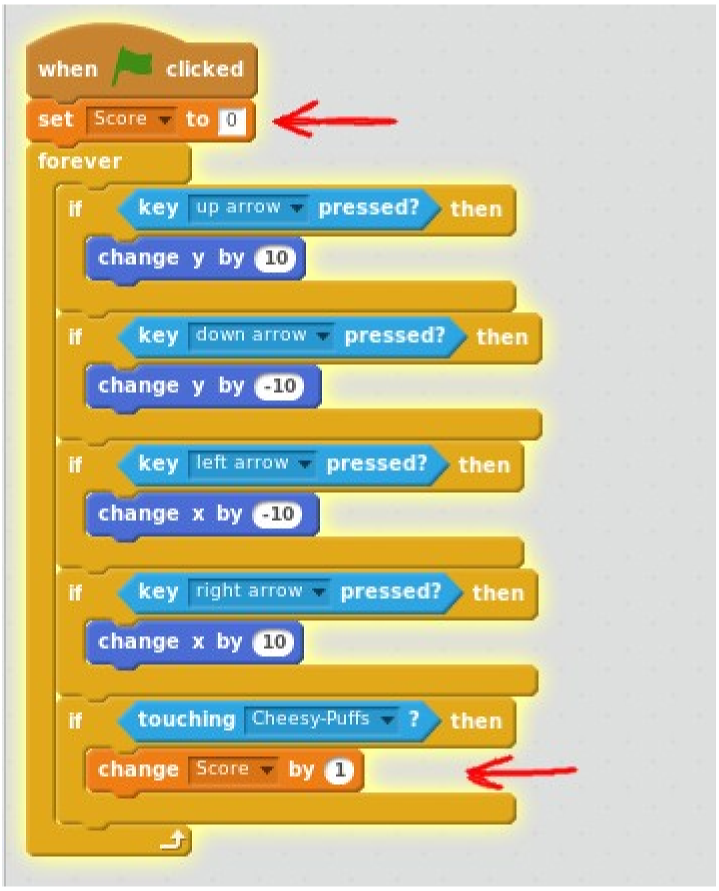
Once you hit “OK”, you'll notice two things: The game screen now has the score drawn at the top-left of the screen (You can move the score box).

Second, we now have a bunch of new Score blocks to play with!

Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

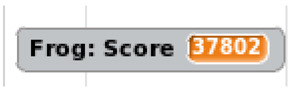
Now, drag the **change Score by 1** block to within our if statement. When the player touches the item, the score will go up by 1!

However, you'll also notice that the score never restarts! We will want to place a **set Score to 0** block right between the **when [green flag] clicked** block and the **forever loop** block.



```
when [green flag] clicked
set Score to 0
forever
    if key up arrow pressed? then
        change y by 10
    if key down arrow pressed? then
        change y by -10
    if key left arrow pressed? then
        change x by -10
    if key right arrow pressed? then
        change x by 10
    if touching Cheesy-Puffs ? then
        change Score by 1
```

Another thing you should notice if you test the game, is that if the Player touches the Item, the item stays in place and the score keeps incrementing until it's a very big number.



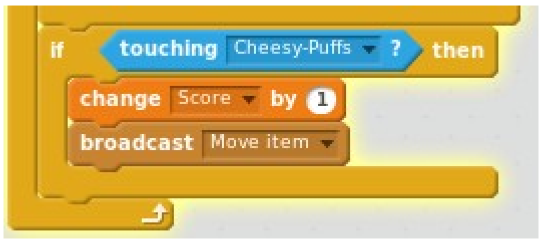
Frog: Score 37802

When the Player retrieves the Item, let's put the Item somewhere else.

However, we're editing the Player sprite right now, not the Item! How can we possibly talk to the Item and tell it to MOVE OUT OF THE WAY?!

We must broadcast a message!

Select the **Events** category. Near the bottom are the broadcast signals. Select **broadcast message1**, then use the dropdown box to create a **new message**. Name it something like, "Move item".



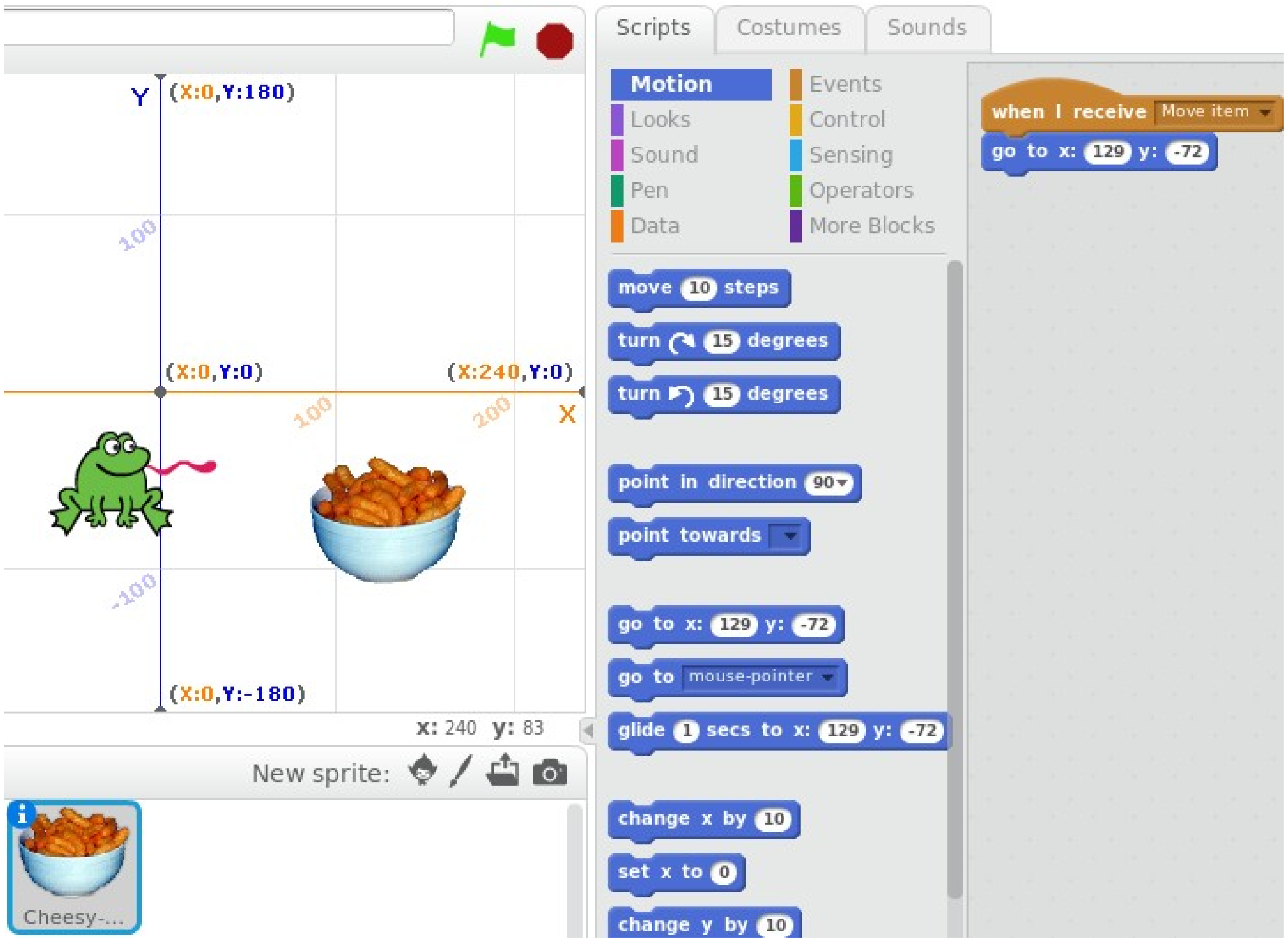
```
if touching Cheesy-Puffs ? then
    change Score by 1
    broadcast Move item
```

We are done with the Player sprite for now. Under the Sprites pane, select your Item. The Scripts pane will clear – there are no scripts for the Item yet, just the Player!

Now, we must add logic to ensure the Item moves when picked up!

Under the **Events** category, select the **when I receive Move item** block and move it into the scripts pane.

Under the **Motion** category, select **go to x: _ y: _**.

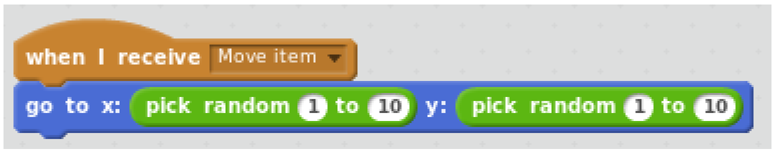


Now, we don't want it to go to the same coordinates every time it's picked up — we want RANDOM.

Go to the **Operators** category. Here, you will see math operations like addition, subtraction, greater than, less than, and more!

Choose the **pick random _ to _** block and move it into both the **x** slot and the **y** slot of our **go to block.**

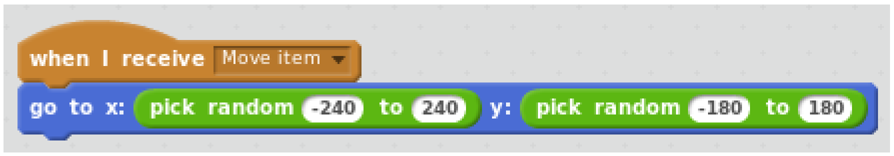
Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/



```
when I receive [Move item ▾]
go to x: (pick random (1) to (10)) y: (pick random (1) to (10))
```

Now, what is the minimum value and maximum value for X and Y? It's on the coordinate grid!

Set the X minimum to -240 and maximum to 240, and set the Y minimum to -180 and maximum to 180.



```
when I receive [Move item ▾]
go to x: (pick random (-240) to (240)) y: (pick random (-180) to (180))
```

We're done!

Press the green flag – we're done making this sample game! You can change the backdrop if you would like. Otherwise, we're able to move the Player, and pick up the Item to increment our score!

Here are some ideas for adding on to this game later:

- When does the game end? Add a condition that, if the player gets a score of 10, they get a "You Win!" screen.
- Have multiple items to pick up, each giving you a different amount of points!
- Play a noise whenever the Player picks up an Item
- Change the player movement so that it uses the **turn _ degrees** Motion block and the **move _ steps** Motion block to move, rather than our **change x** and **change y** blocks

Licensed under CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/