# Review:

- Draw a line
- Join a String
- Sprite Heading and speed
- Increment a variable
- Top of the canvas
- Object type (variables)
- Loops


Introduction to App Development: Lesson 5

# Practice & Functions.

# Code can be tedious sometimes…

- Today we're going to see how we can make coding easier for ourselves.

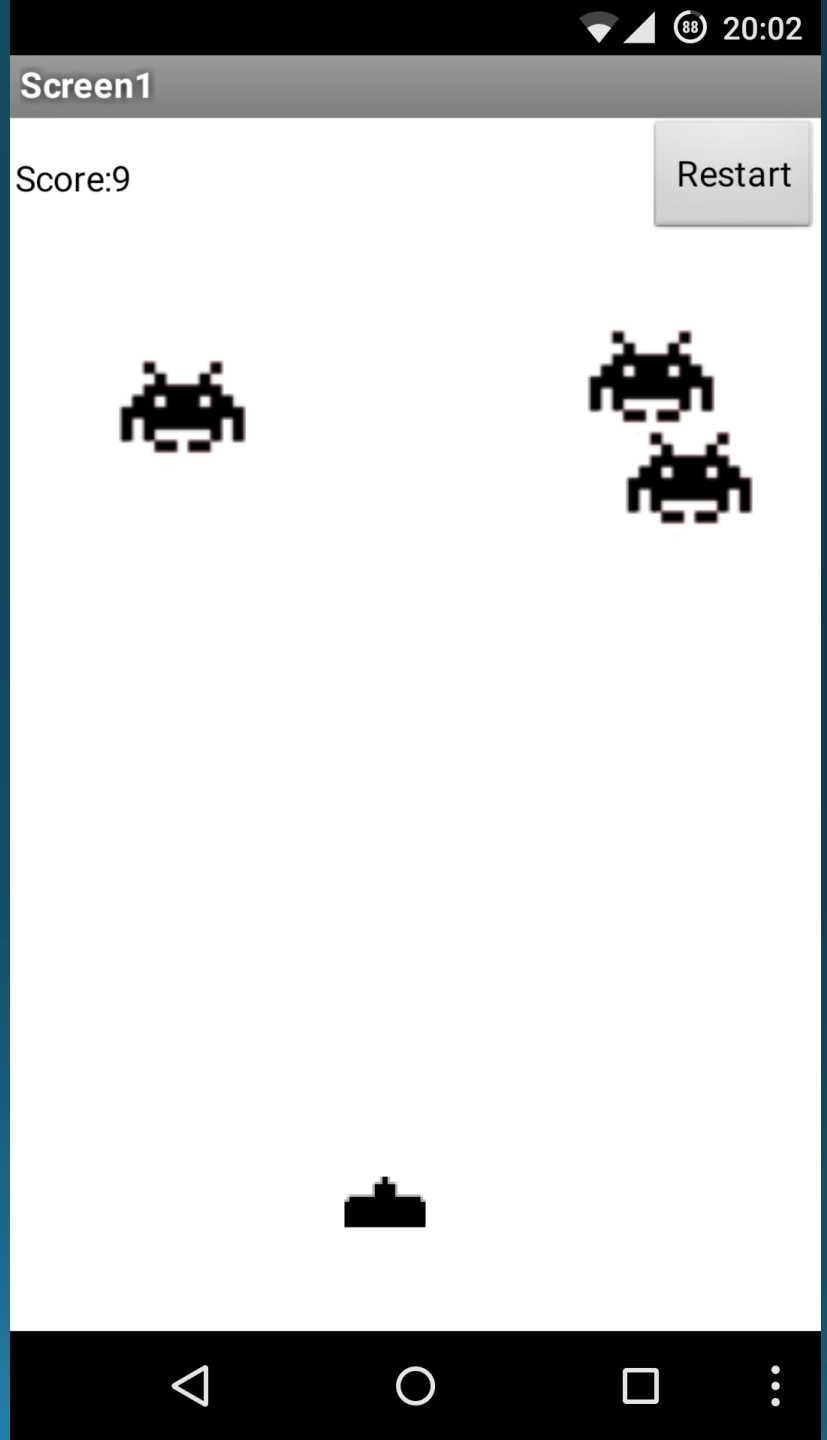
- We will learn how to make reusable and dynamic code so that we spend less time dragging blocks and more time coming up with great ideas.

# The first part of today's lecture is practice

- We will start by making an app that uses skills we already know.
- Before showing the code for the app, let's talk about the functionality
- Try to produce the logic before we show it in the slides.

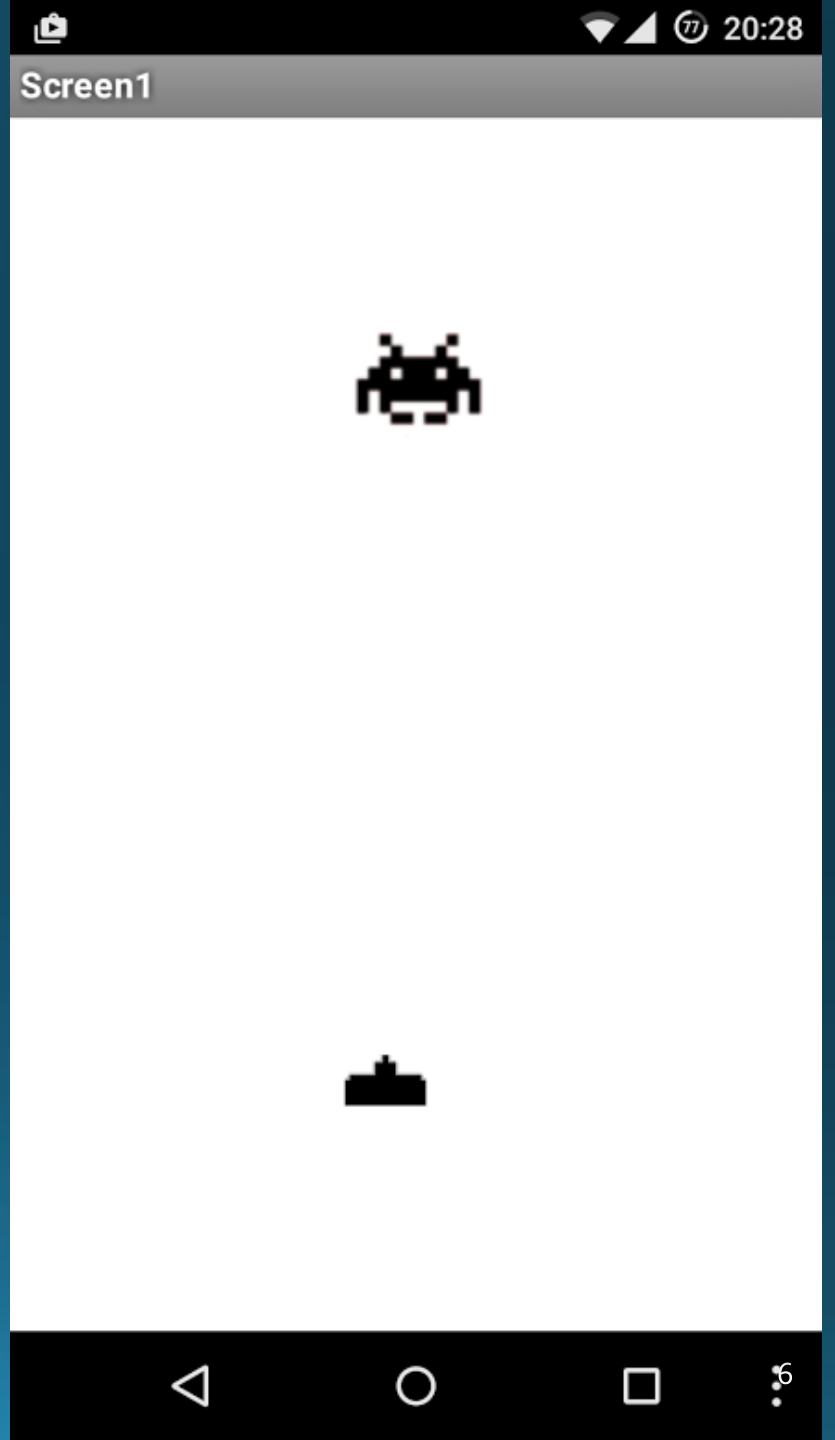
# Today's App: AlienIntruders

- Make a new project!
- Upload the unzipped media files from the Course website:
  - caltoc.scs.ryerson.ca



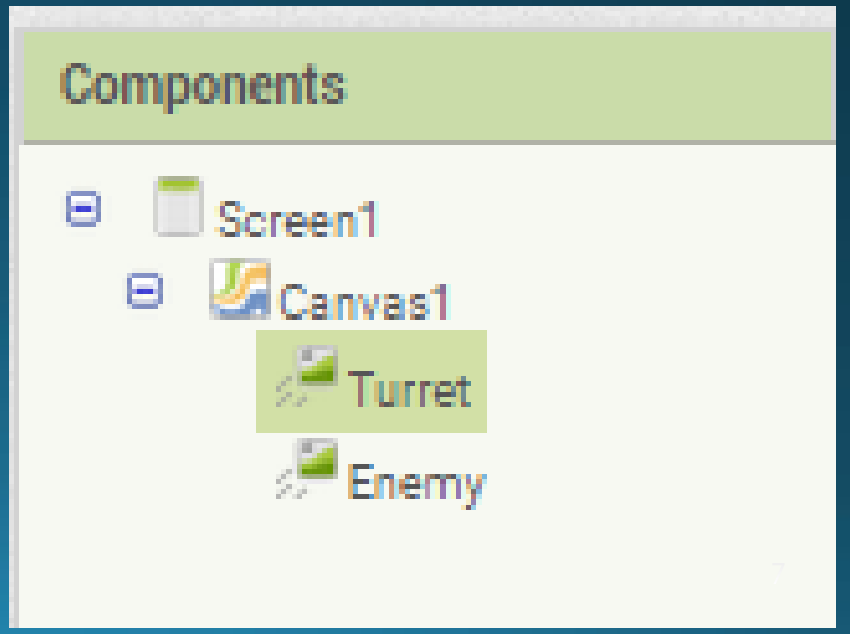
# Let's make a game!

- In this game, Enemies fall straight down; if they reach the bottom, the player loses.
- We'll start with one player and one enemy.
- What 3 components do we need?



# Components:

- Set up the following components:
  - **Canvas**
    - Set the Width to Fill Parent
    - Set the Height to Fill Parent
  - **ImageSprite (Turret)**
    - Set the Picture to player.png
    - Move it to the bottom of the screen
    - Rename it to Turret
  - **Another Imagesprite (Enemy)**
    - Set the picture to enemy.png
    - Move it to the top of the screen
    - Rename it to Enemy

# Make it fall

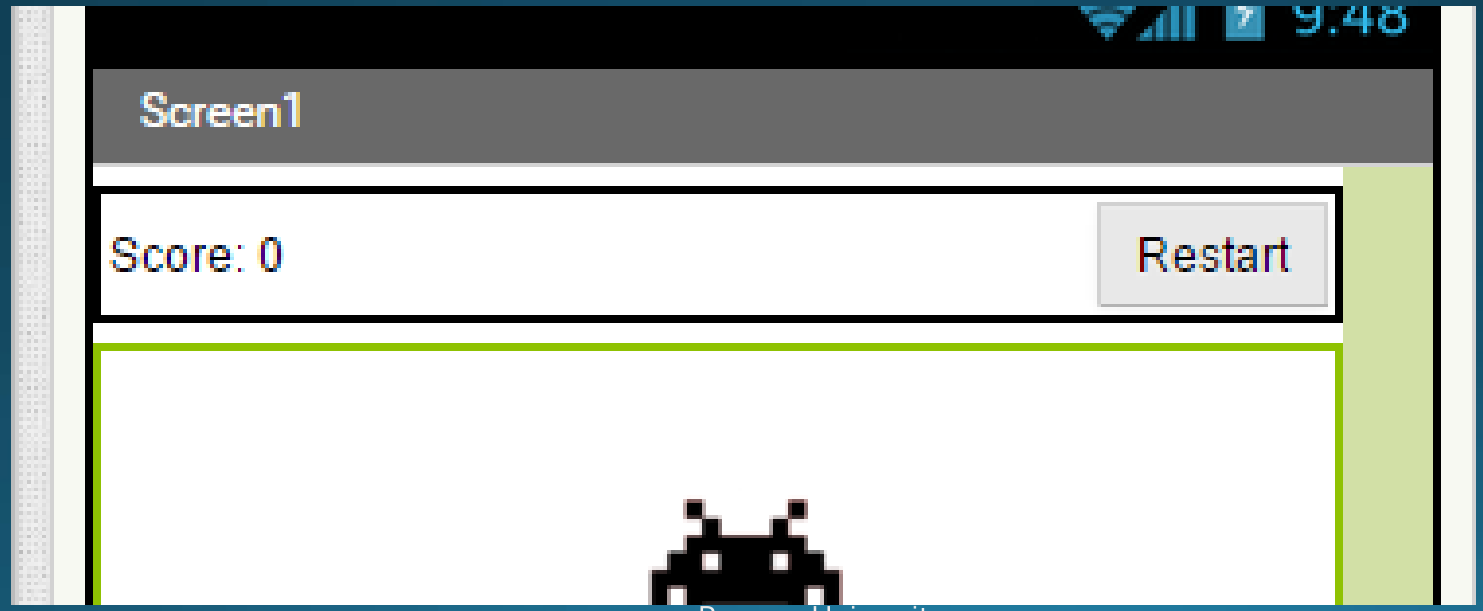
- We can make the Enemy fall without using any code

- Update the following properties on the Enemy ImageSprite to make it fall
  - Heading
  - Speed
  - Interval
  - Rotates

# Keeping Score

- Add a bar at the top for showing the user's score and allowing them to restart the game

- You'll have to use:
  - A horizontal arrangement
  - A Label
  - A Button
- Set the text and width properties of each accordingly.



# Here's the plan:

- When enemy is touched (use TouchDown):

    1. Draw a line from the Turret to the Enemy
       (so it appears that we shot it)

    2. Move the Enemy to a random place at the top of the canvas
       (so it appears that it died and respawned)

    3. Increment the score

    4. Update the label's text

**Try to build these blocks before we move on.**


- Here is the completed logic
- Did we forget anything?
- Copy any missing logic before we move on

Test your app.

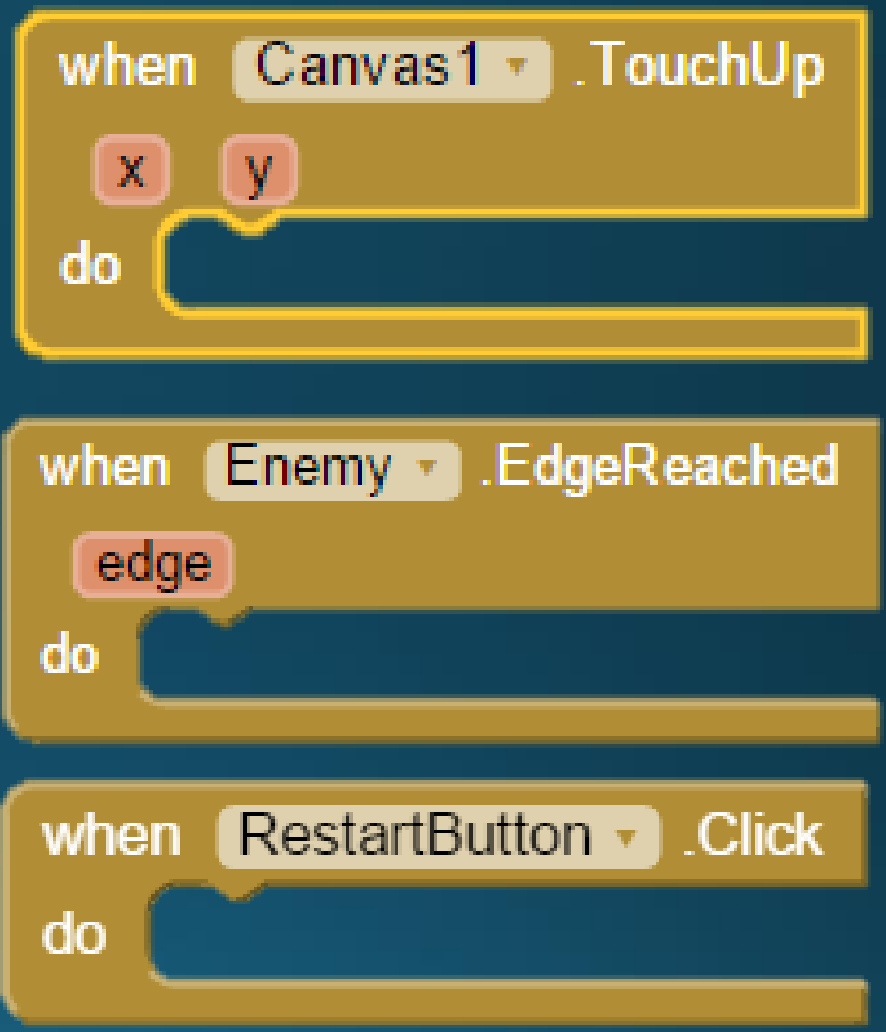
# What's Wrong?

- Ugly lines
- No lose condition (user can't lose)
- Button doesn't work.

Any ideas how to fix this?

# Three more events:

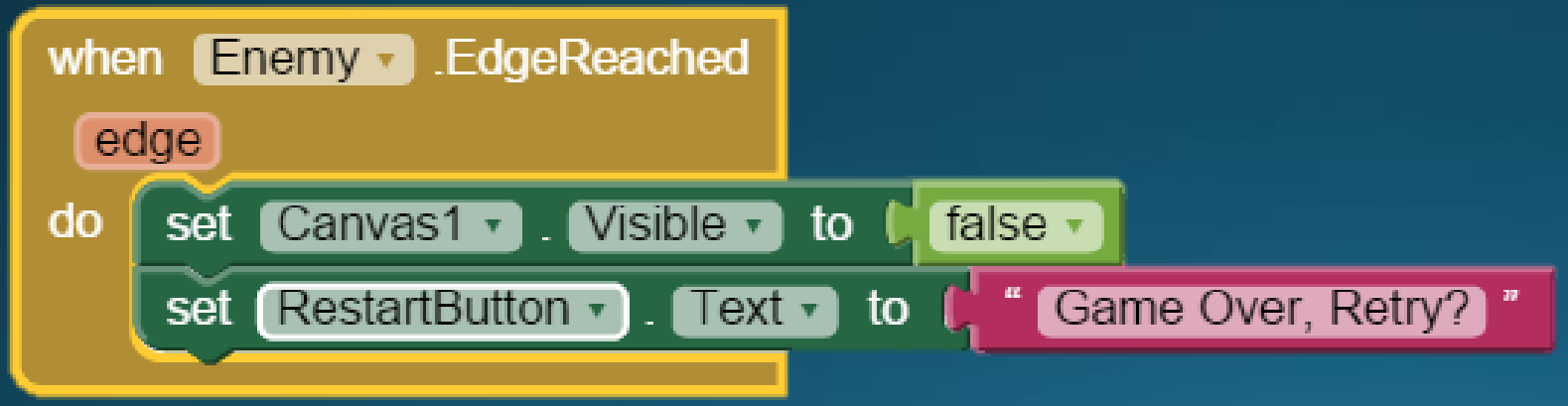
- **When the user lifts their finger:**
  - Clear the canvas
- **When the enemy reaches the edge**
  - Disappear the Canvas
  - Change button to say: GameOver, Retry?
- **When the user clicks restart:**
  - Reappear the Canvas
  - Move the Enemy to a random place at the top of the canvas
  - Set Score to 0 and Update the label
  - Change button to say: Restart

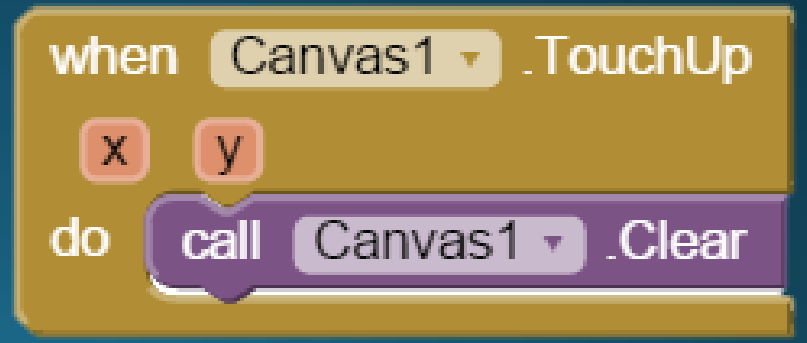



Give it a try!

# Completed logic


- Did we forget anything?
- Copy any missing logic before we move on.



Test your app.

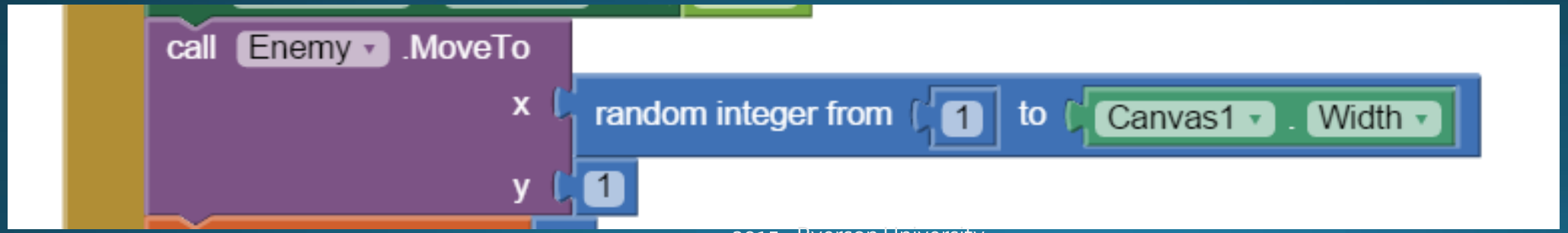
# What's Wrong?

- The game is ending randomly!

# Can you tell why?

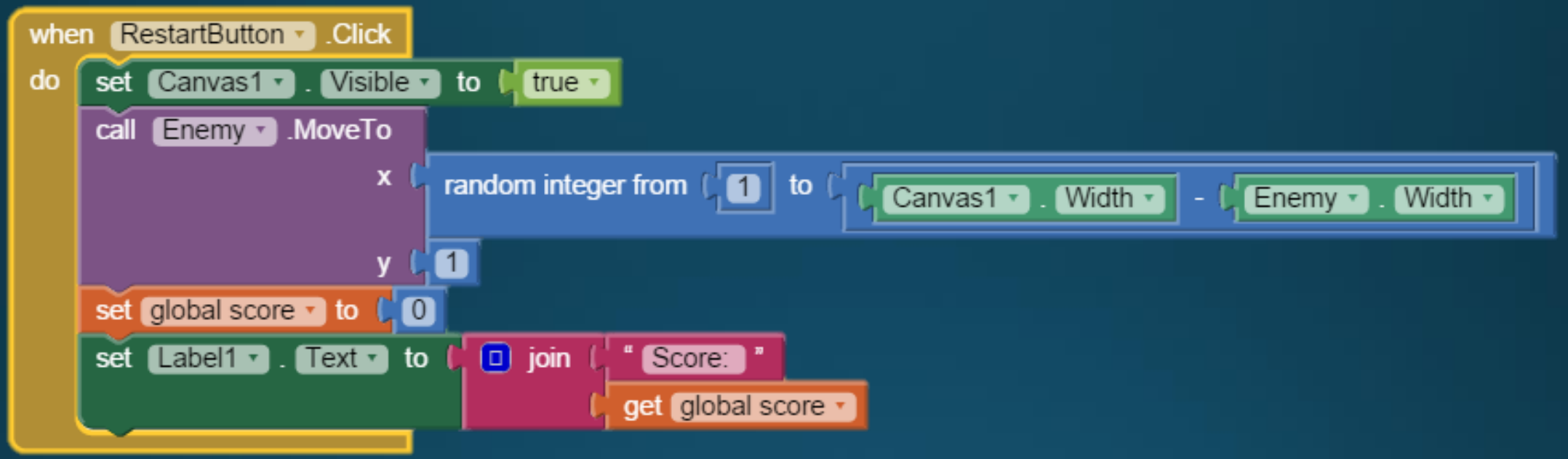
# Creepy crawlies

- There's a bug, the Enemy.OnEdgeReached event is happening randomly!
- When we move the enemy, sometimes we crash him into the right edge.
- **Fix the bug, It's in here:**



# Did you fix both occurrences?

# Bugfix




- So now we have fixed the bug.
- But how many times did we have to change it?
- Doesn't this MoveTo block look large and complicated?
  - Not only that, but it shows up twice in our code!

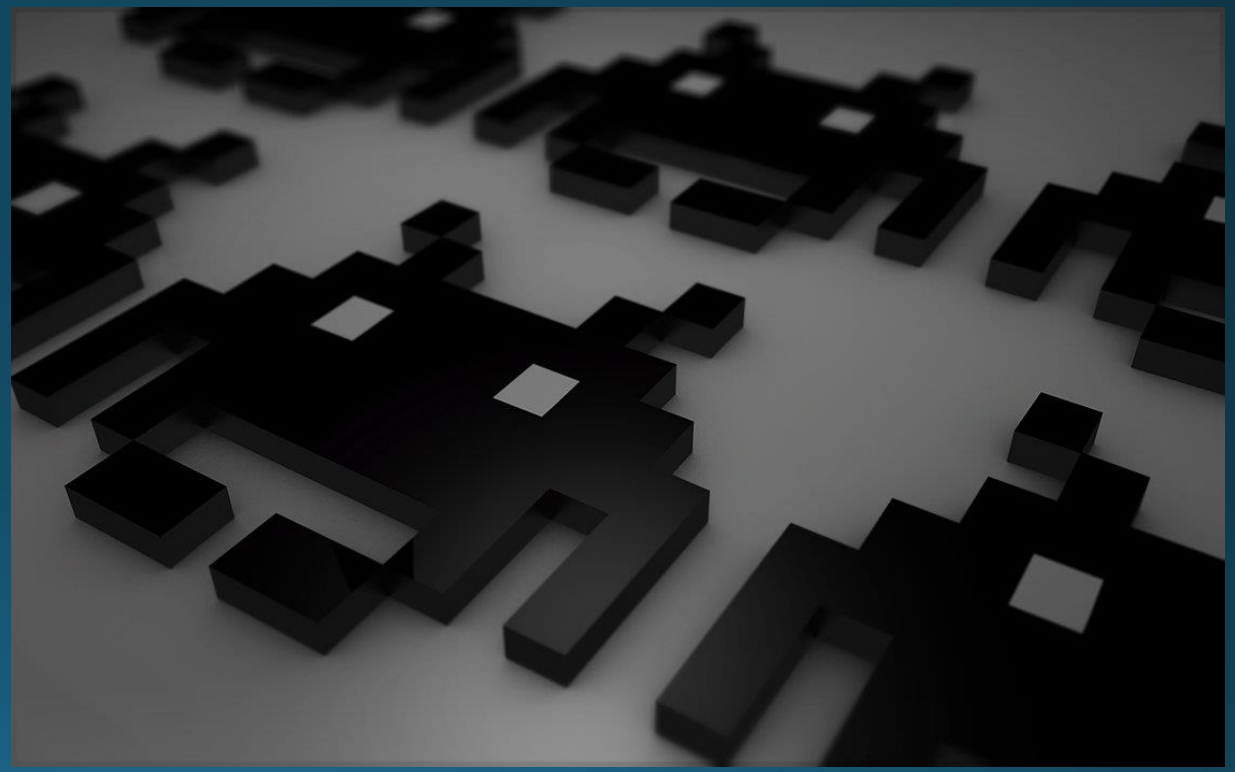
# Alternative…

- Fortunately, we can simplify our code a bit
- Replace that MoveTo block with a RepositionAlien block
- **Raise your hand if we can't find it.**


Doesn't this code look clearer?

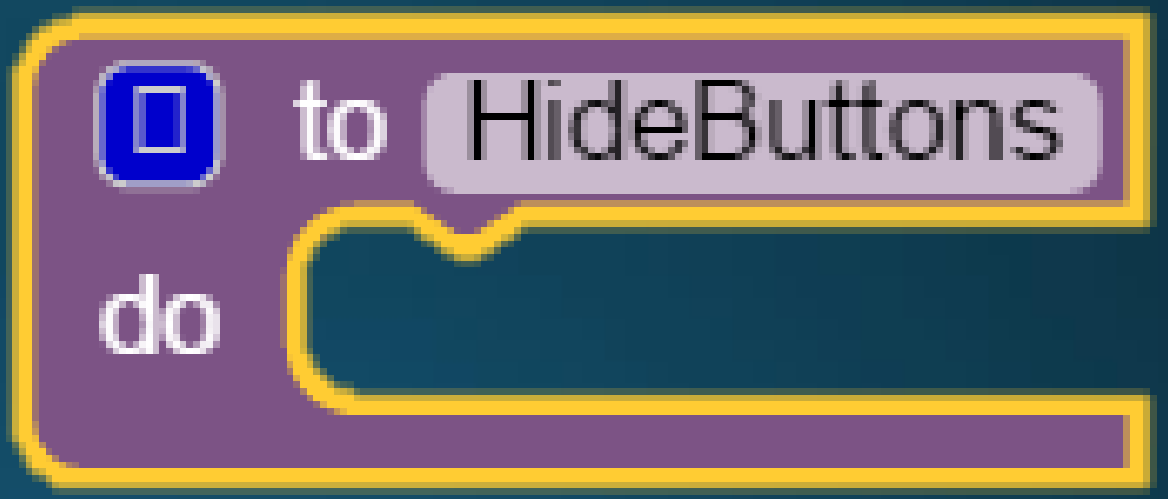
# Creating new blocks!

- Well, of course we can't find the block.
- There is no repositionAlien block in AppInventor.
- repositionAlien is a procedure we want to execute.



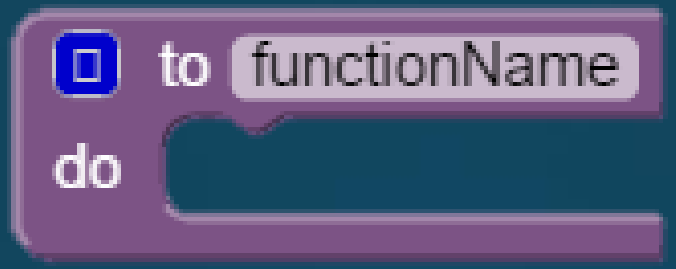
# Functions! (AKA Procedures) Concept #10

- Functions let us group together blocks and reuse them many times in different places across our code.




- Functions have two parts:
  1. Function definition
  2. Function Call

# Function definition



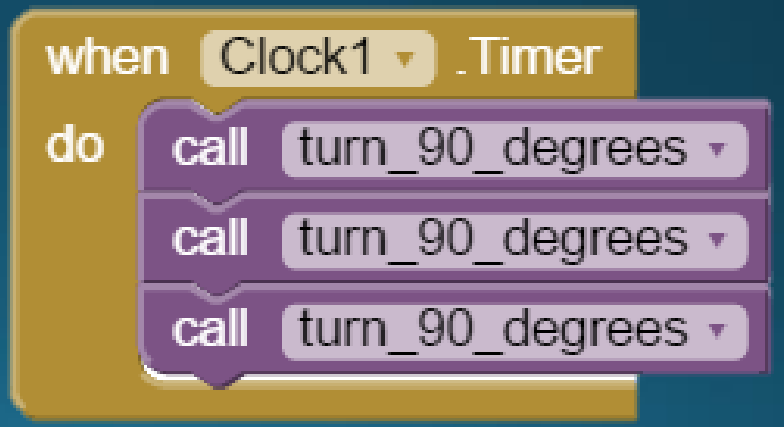

- A function definition contains all of the blocks that will be executed when the function is called.
- The blocks won't execute by themselves: the function must first be called elsewhere.

# Function call

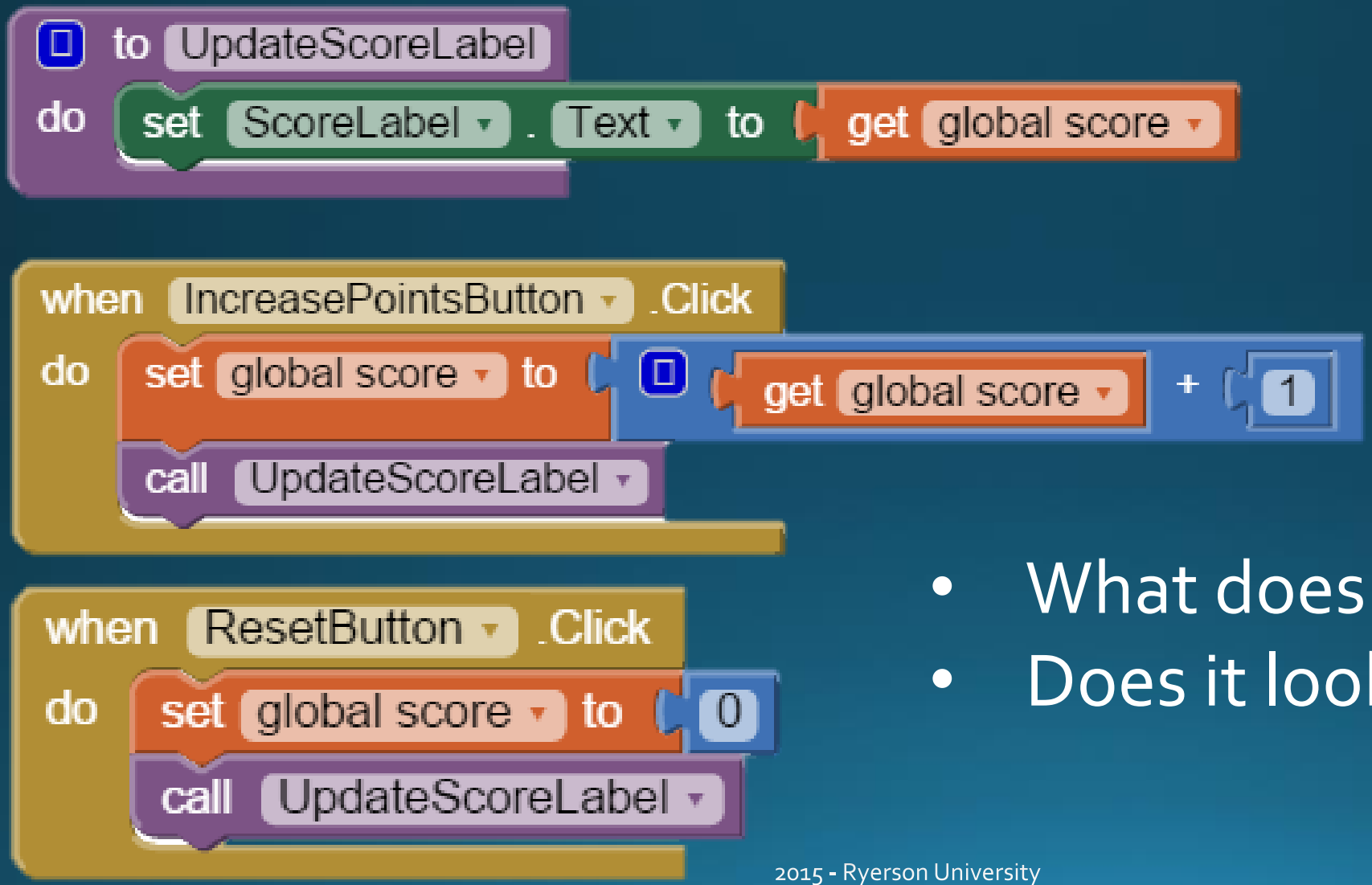

- We saw these in the first lecture!
- A function call executes the function (and all of the blocks inside it's corresponding definition).
- After the definition is built, we can call the function many times easily anywhere we want!

# Function Example #1


# What does this code do?



# Function Example #2



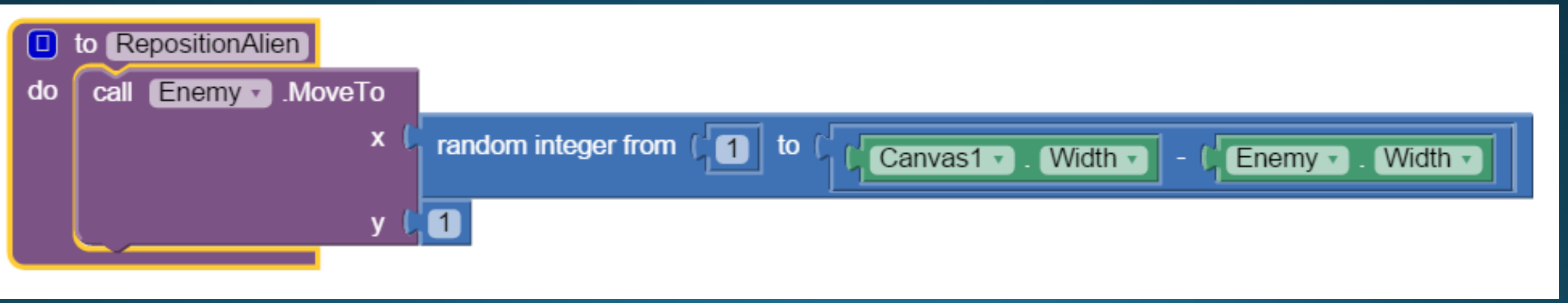

- What does this code do?
- Does it look familiar?

# Let's get back to our App

Earth is in danger.....

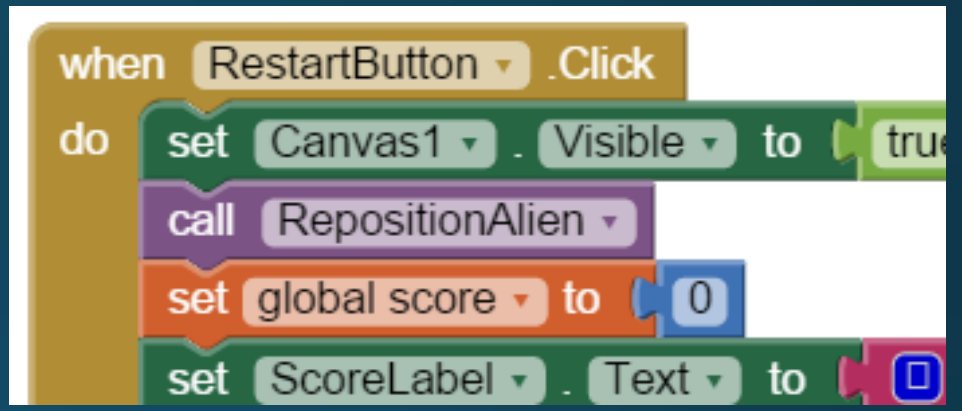
- From the Procedures menu in the Blocks view:
  - drag in a procedure block


- Change it's name to RepositionAlien and drag the MoveTo logic into it.

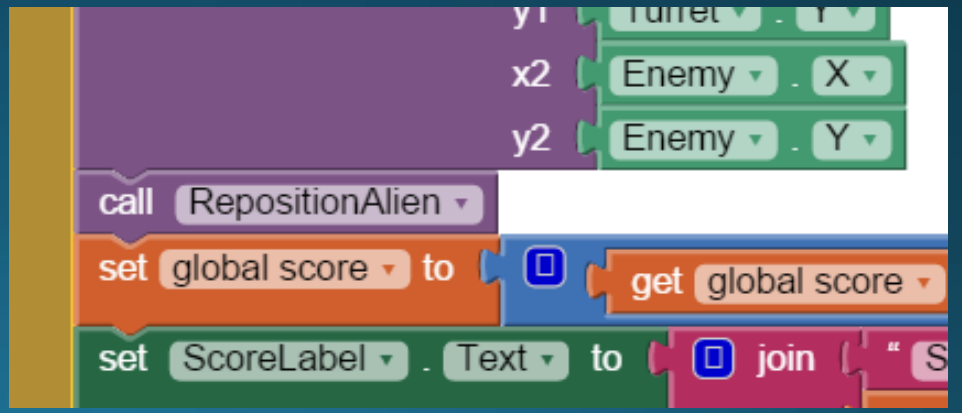


# Using the RepositionAlien block.

- From the Procedures menu, we will now find a block called **repositionAlien**




- Drag the block out and replace the **MoveTo** block in both places.



# Test your app, is it still the same?

If so, not to worry, we'll be right back with more functions.

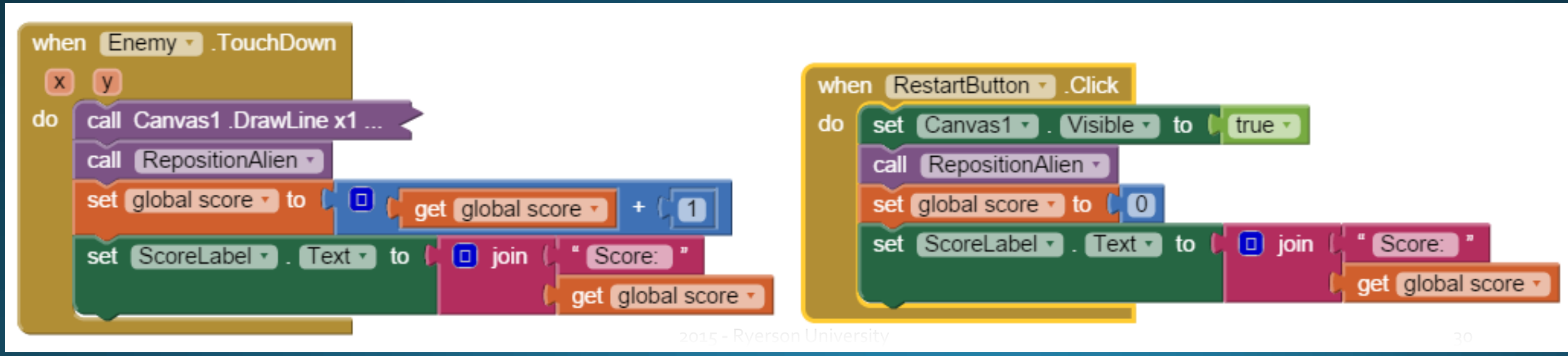
# Code Repetition

Code repetition is bad:

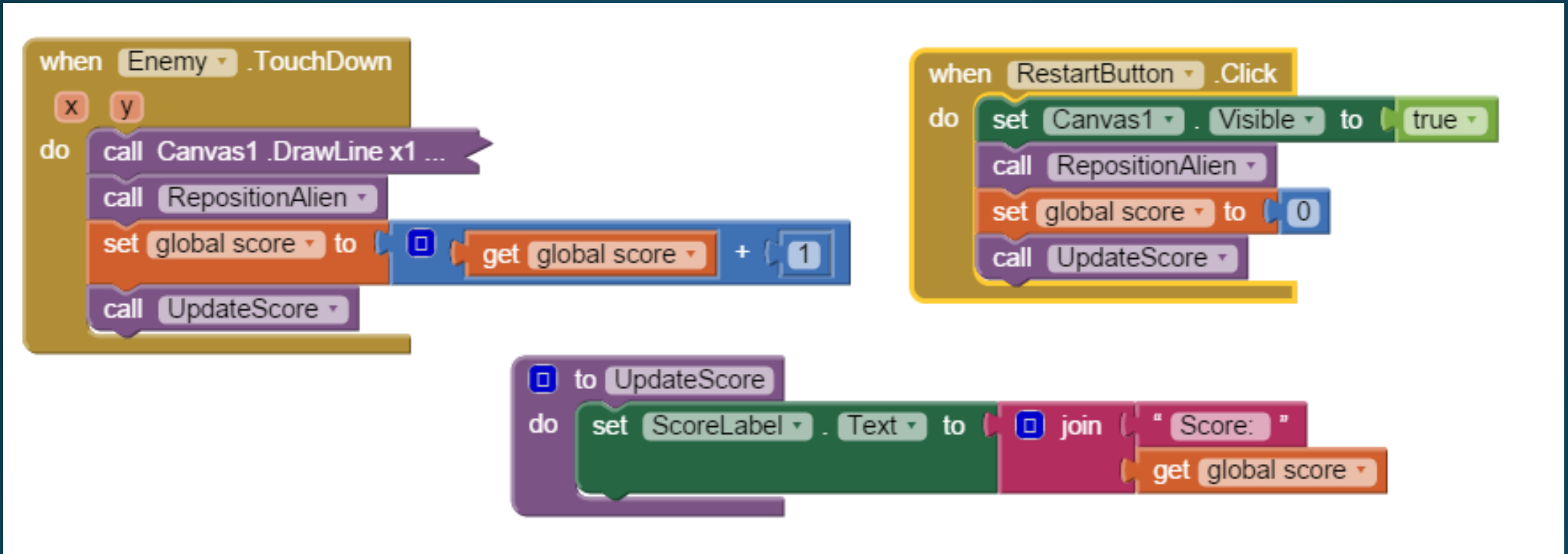
- As we saw earlier, if we fix a bug in one line of code, we might forget to fix it in other places where it may exist.
- Removing repetition makes our code easier to understand.
- Do we see any other parts of our program where code is repeated?

# Repetition

- What blocks do we have here that we could put in a function
- Could we make an updateScore function?
- **Go ahead and give it a try.**



# Considerations




- Hmm, this is good… but can we do better?
- Can we add the  block to the function?

- No. The value we set the score to is different in each case.
- However….

# Function Parameters

- Functions can take parameters:
  - We covered these when we explained function calls in the first lecture!

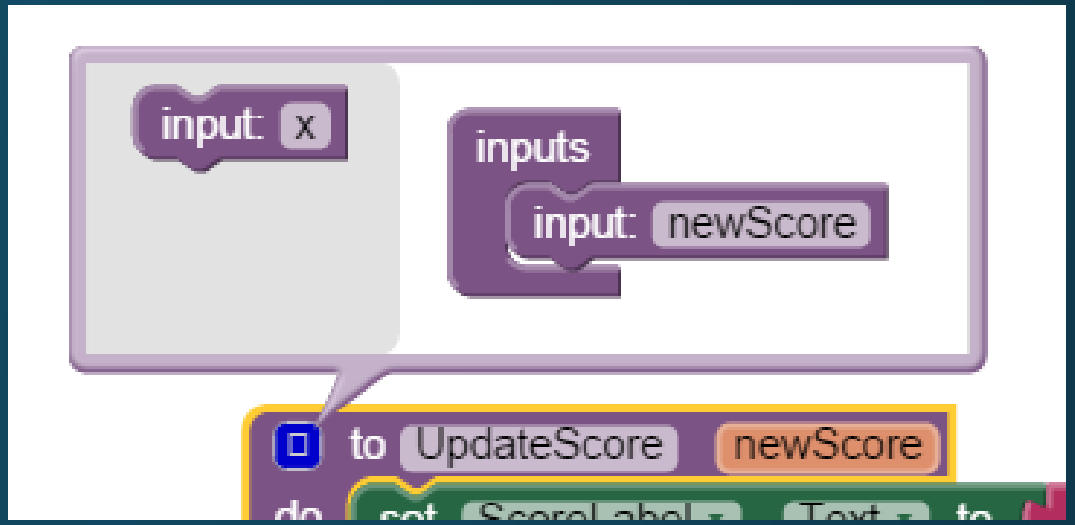

- Basically, we can make a function that asks the programmer (us) for some information


What does this code do?

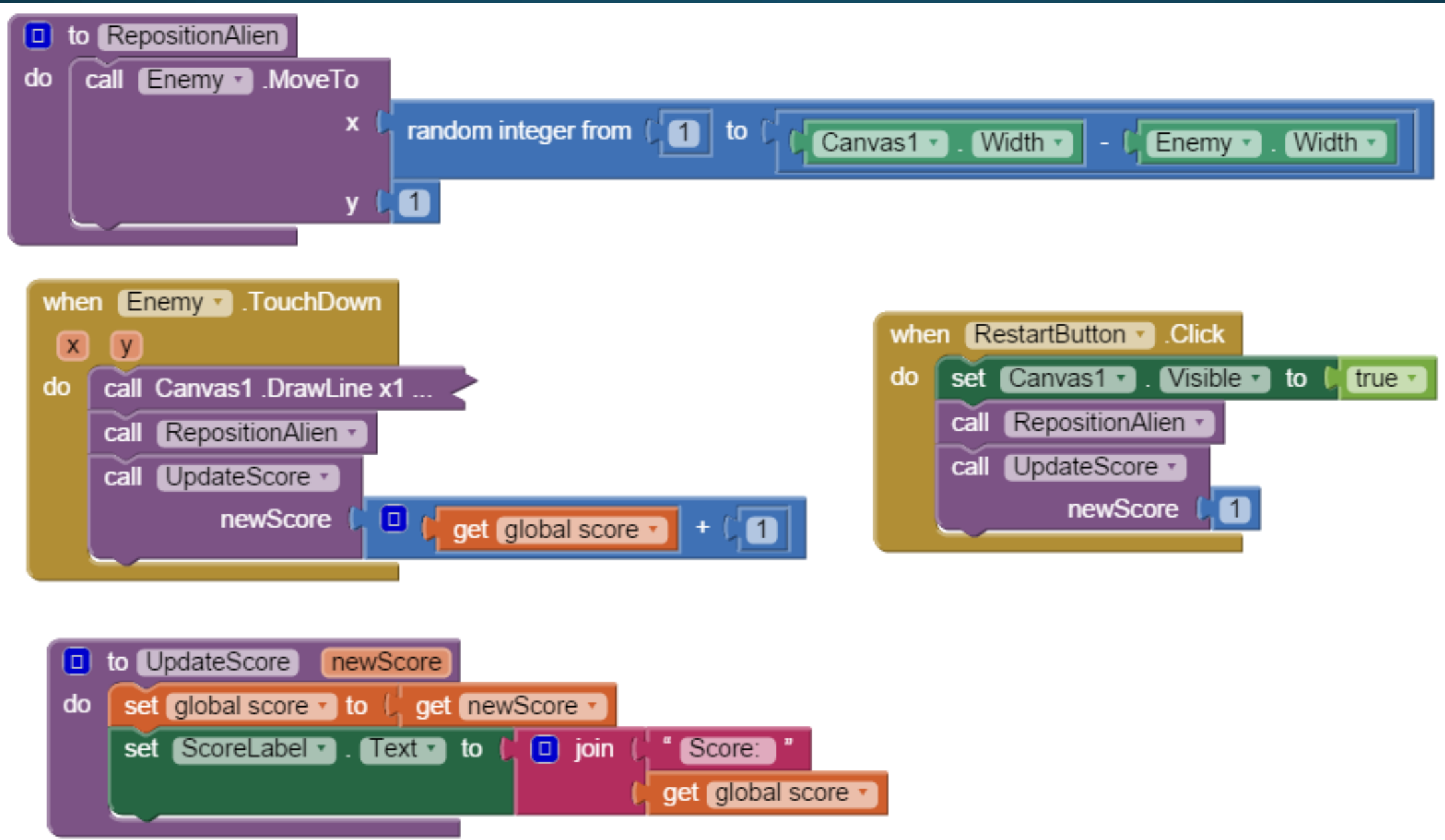
Pretty easy to read, right?

# Inputs

- Use the little blue box to add a parameter to the function.

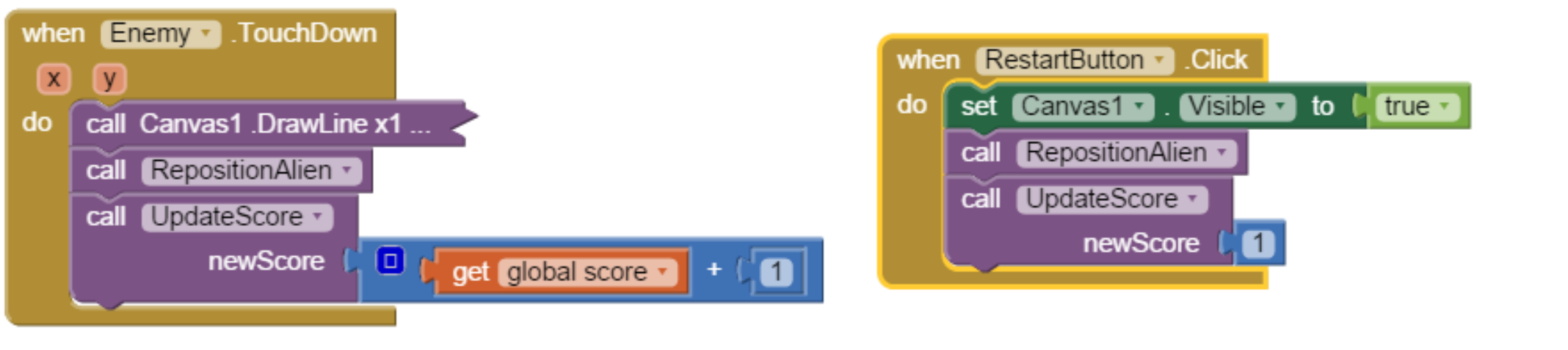



- Update the UpdateScore function so that it also changes the global Score Variable


to RepositionAlien
do
call Enemy .MoveTo
x
random integer from 1 to Canvas1 . Width - Enemy . Width
y 1
when Enemy .TouchDown
x y
do
call Canvas1 .DrawLine x1 ...
call RepositionAlien
call UpdateScore
newScore
get global score + 1
when RestartButton .Click
do
set Canvas1 . Visible to true
call RepositionAlien
call UpdateScore
newScore 1
to UpdateScore newScore
do
set global score to get newScore
set ScoreLabel . Text to join " Score: "
get global score

# An important benefit

- Now that we've made these two functions, let's look at our events:

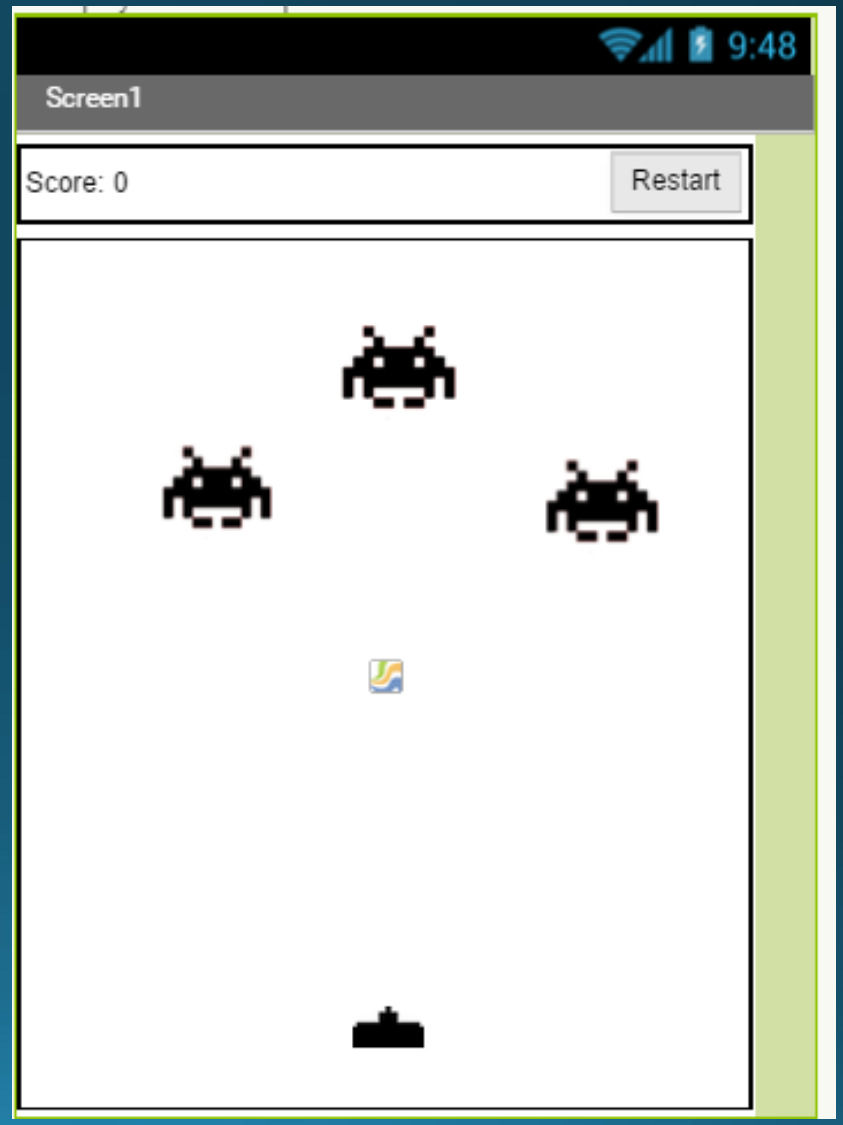



- When we read these events, it is very clear what they do.
- The reason is that we're focusing on what happens, not how it happens.
- This is called Abstraction and is essential for making more complex software.

# Next Step: Add more Intruders

- Let's go back into the designer and add a couple more image sprites.

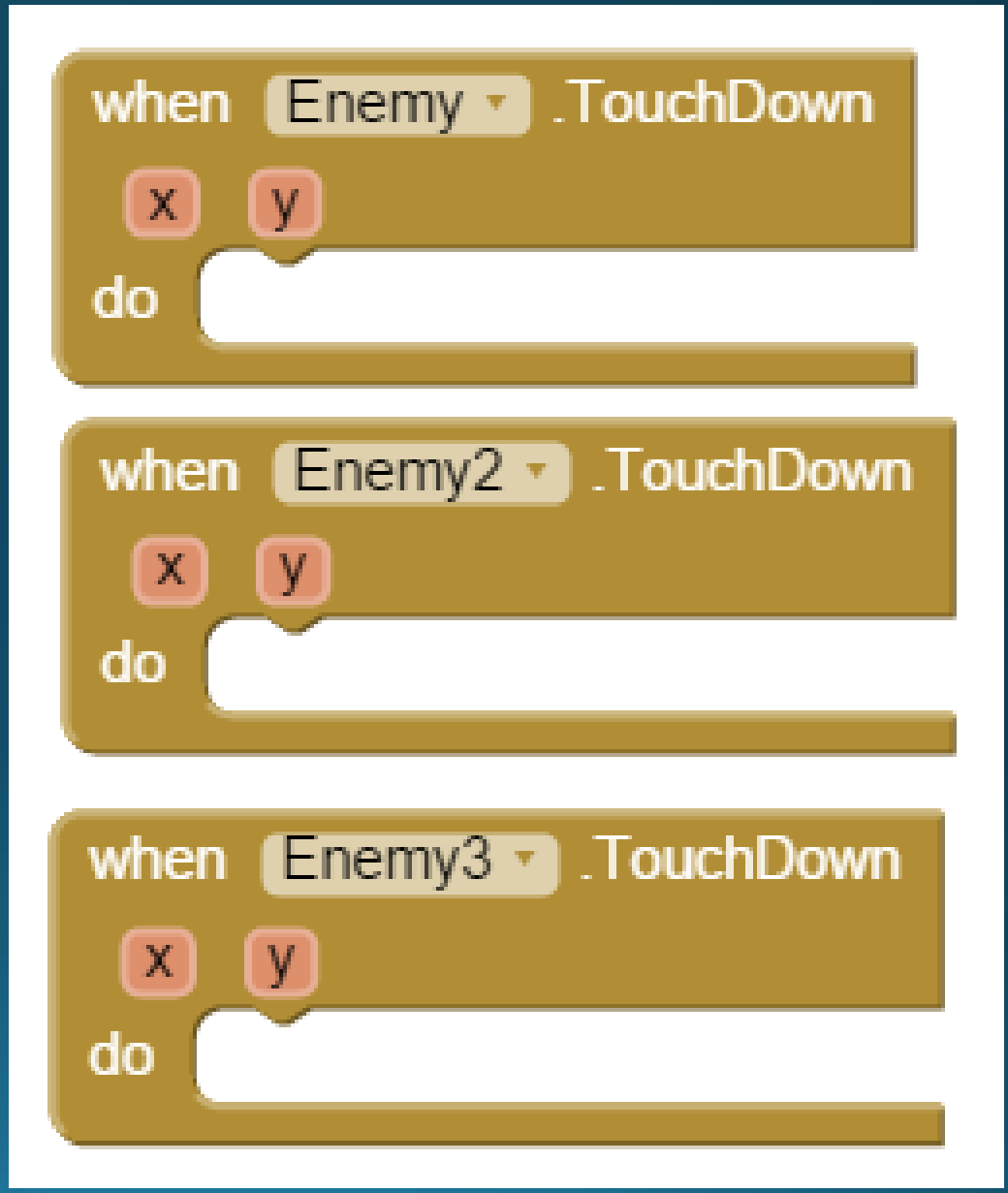
- Set their properties to match the first Enemy:
  - Heading
  - Speed
  - Interval
  - Rotates
  - Picture
- Name them Enemy2 and Enemy3
- Rename Enemy to Enemy1



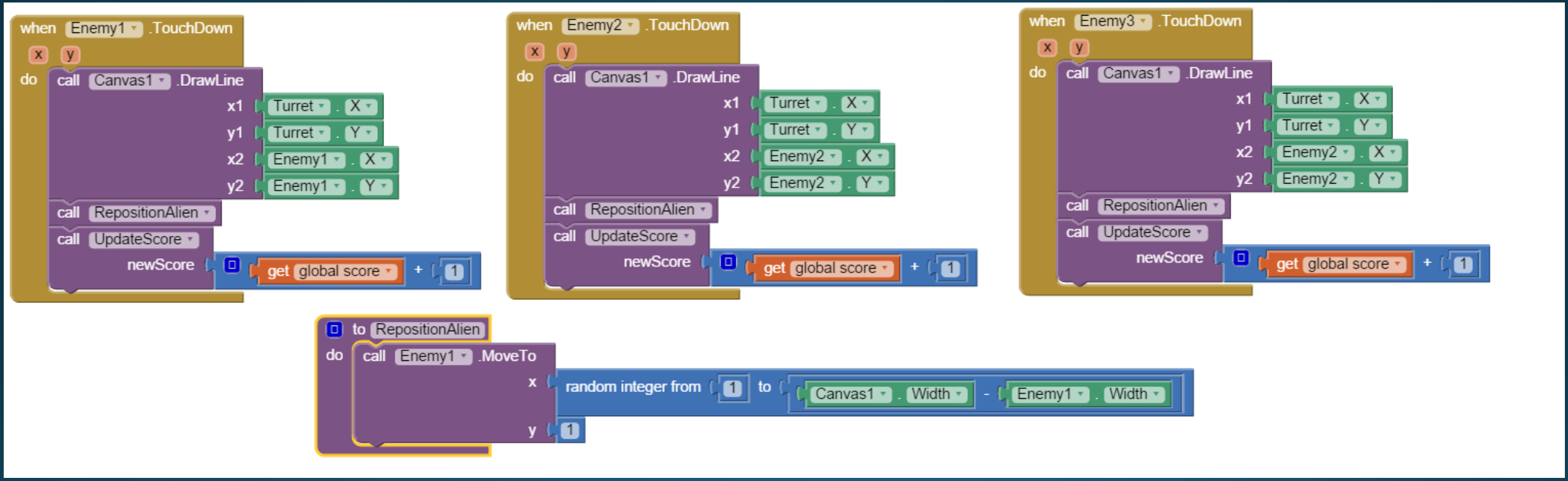
# Adding logic:

- Now we have three TouchDown events to write:

- What do you think this would look like?



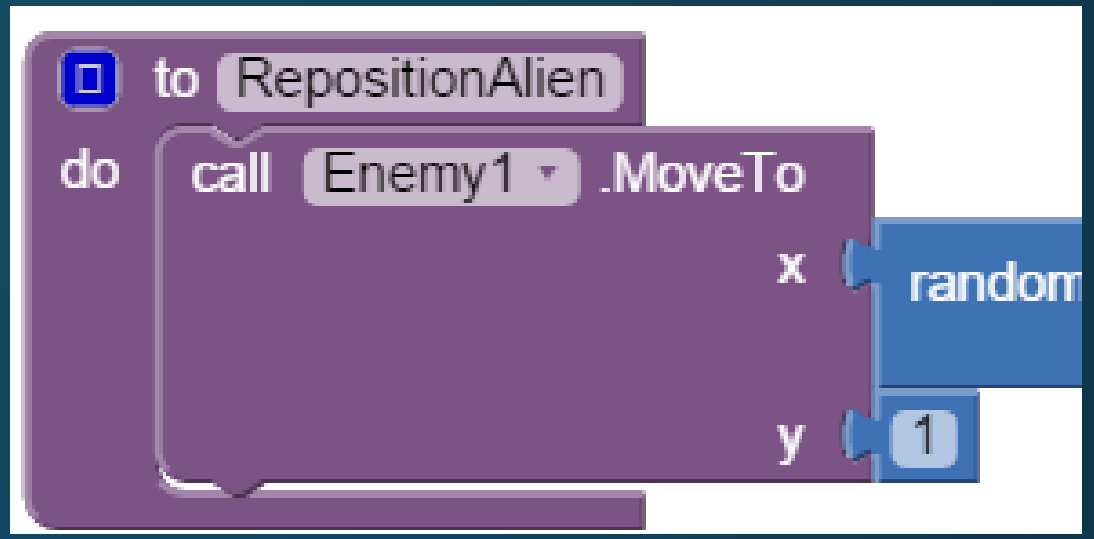
# What it would look like…




Well, this isn't that bad, except there's a big problem….

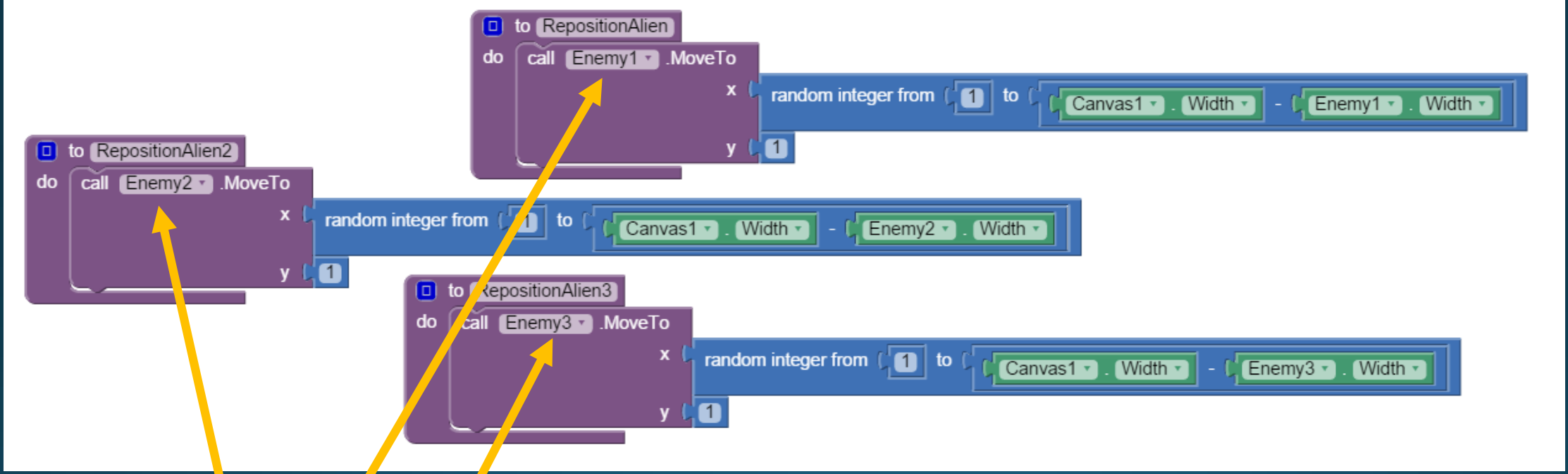
# Let's zoom in…

- This function gets called by
  - **Enemy1.TouchDown**
  - **Enemy2.TouchDown**
  - **Enemy3.TouchDown**



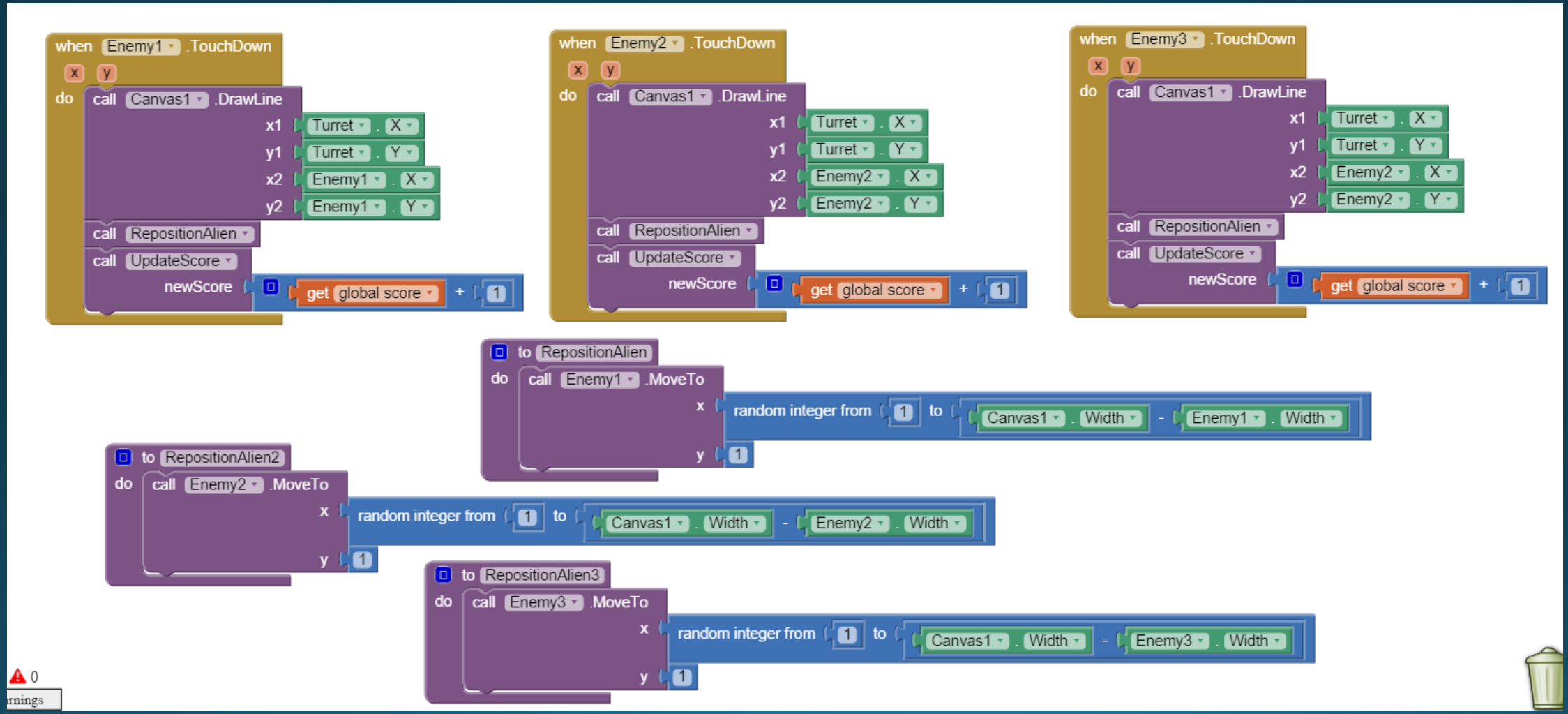

- Can you see the problem?
- What happens when Enemy3 gets touched?
- How do we solve this?

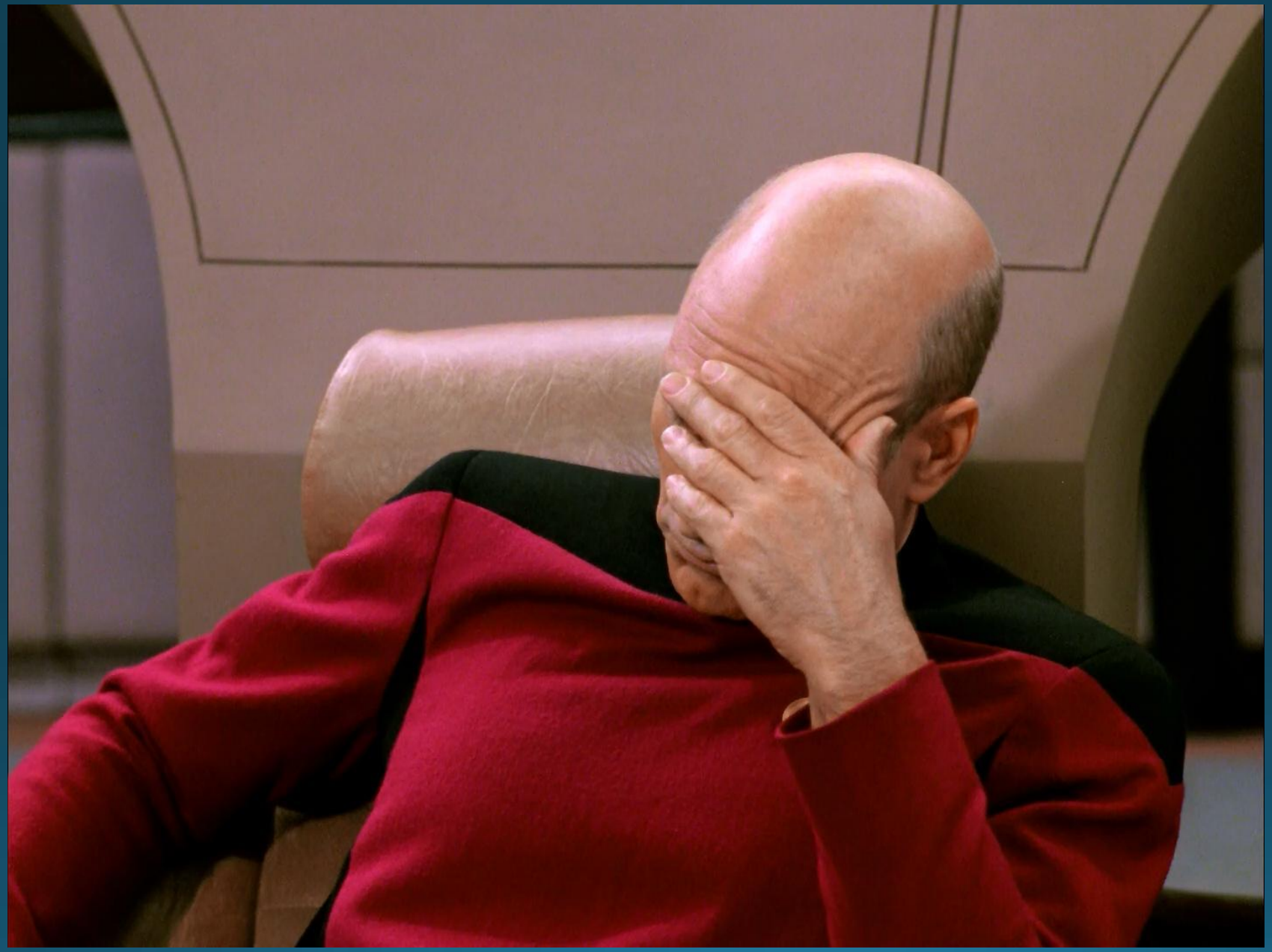
# The solution?




Ok, so we use three functions. Each one uses a different enemy. but that means……

# Just the TouchDown event…





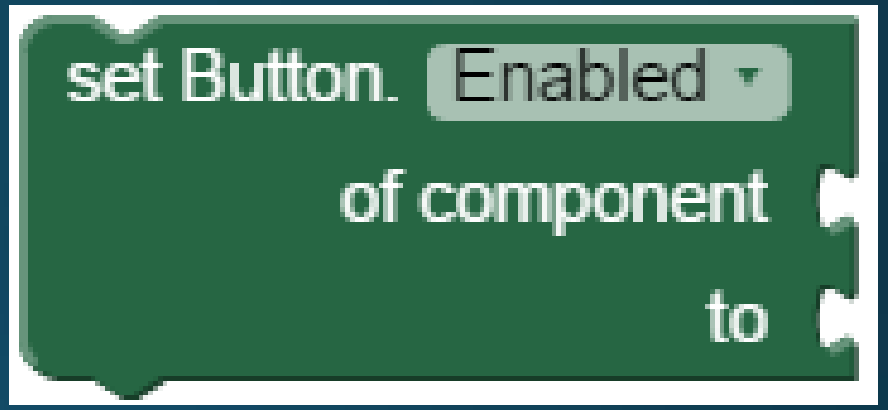
# OK, enough. How do we actually do it in less than 6 hours?

- In AppInventor, blocks have the component built-in.
- The problem is that we want to change the component that we use the block of.
- In order to do that, we'll have to use dynamic blocks
  - AKA "Any" Blocks

# Dynamic Blocks! (AKA "Any" blocks)

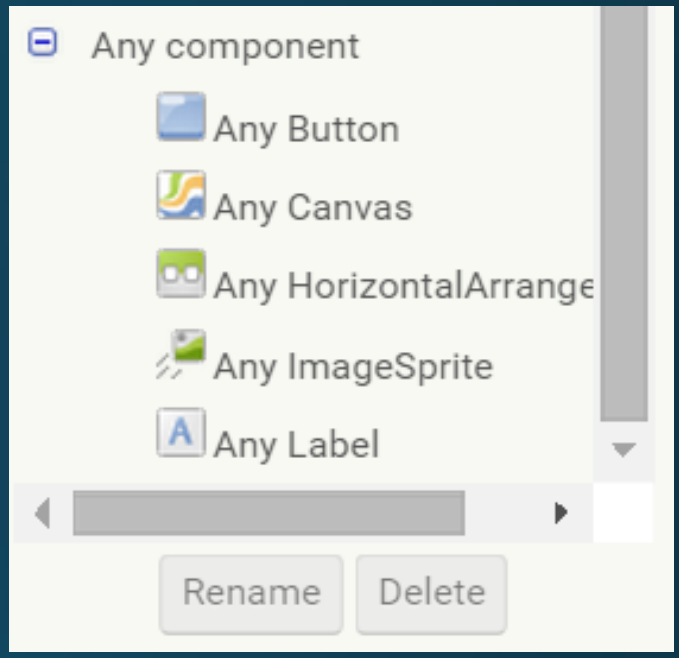
Concept #11

- These have exactly the same functionality as the blocks you've been using so far.



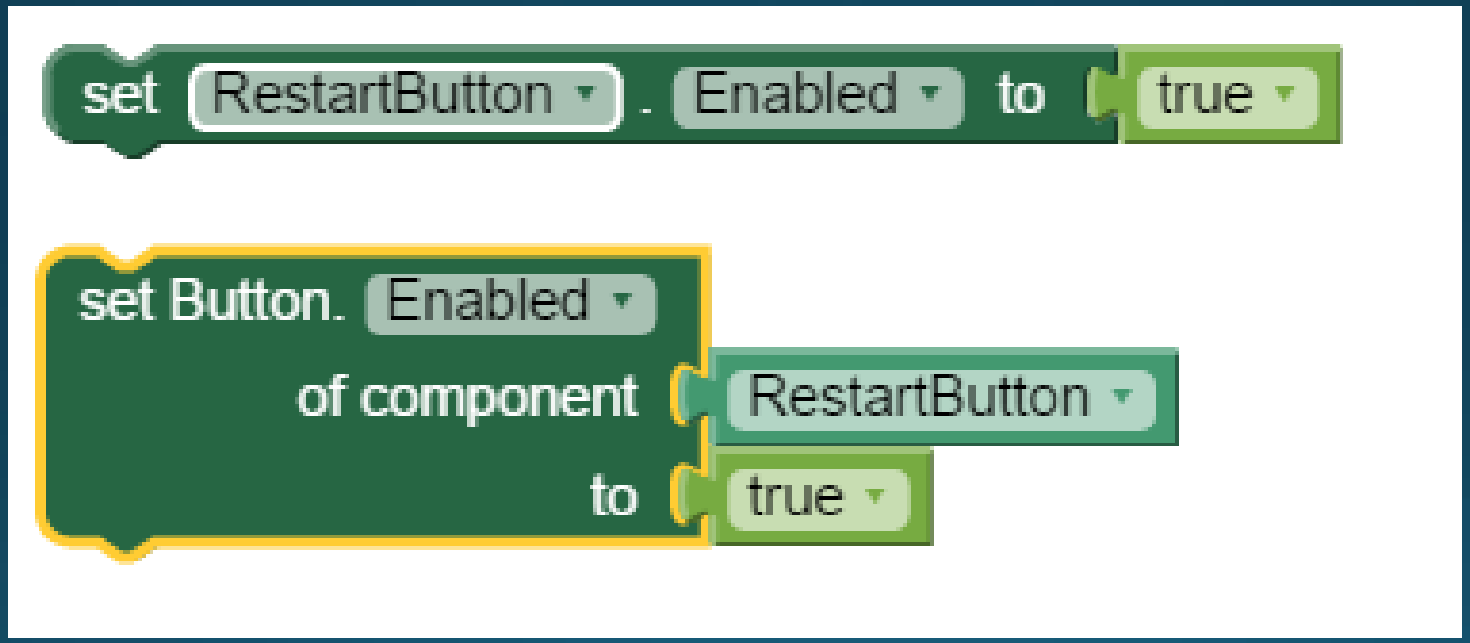

- The difference is that they have an additional parameter for what component they're working on.

# How to use dynamic blocks?




- Dynamic blocks are kept under the Any component menu, and are sorted by the type of component that they refer to.
- To use them, we have to use values of object type.
- To find these, click on a component (eg, Enemy1) and scroll all the way to the bottom.

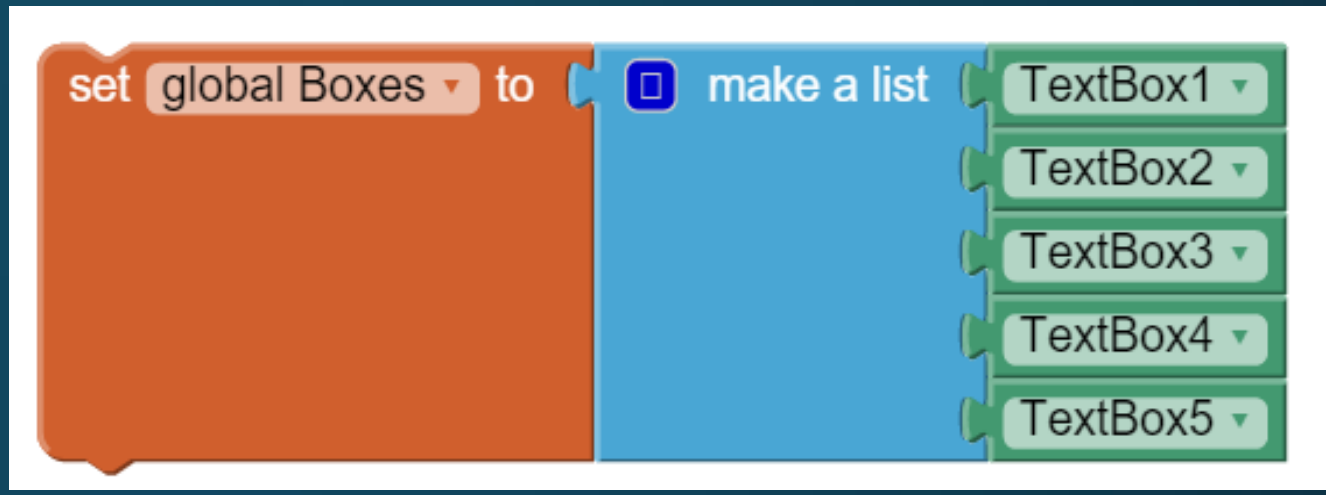
# What's the difference between these two blocks?

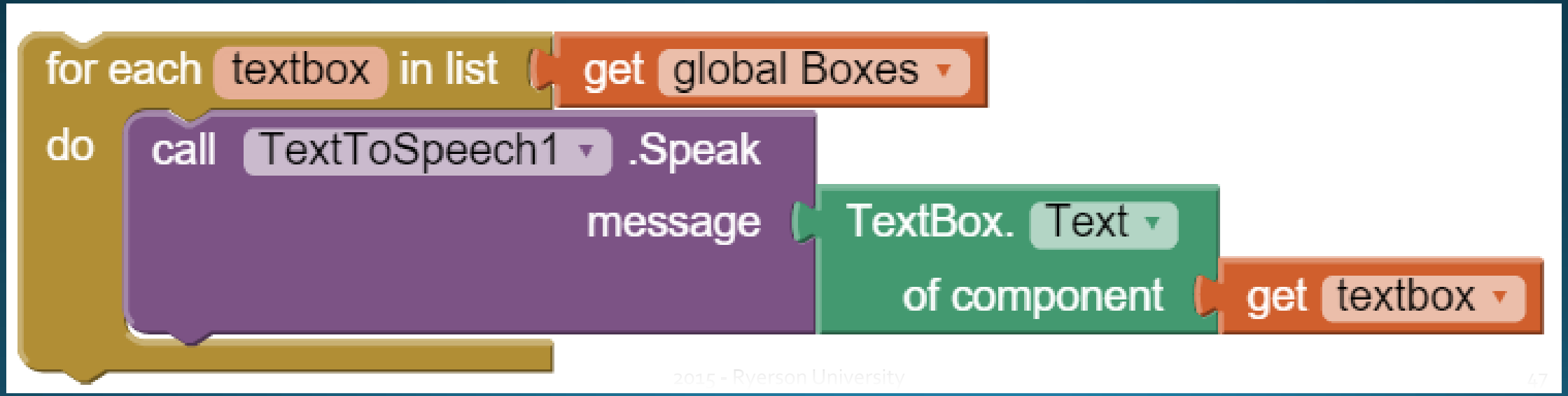



- **Nothing. These blocks both do the same thing.**

# Why use dynamic blocks?

- Take a look at this logic:
- Identify the dynamic block
- What does it do?
- Can we do it without dynamic blocks?

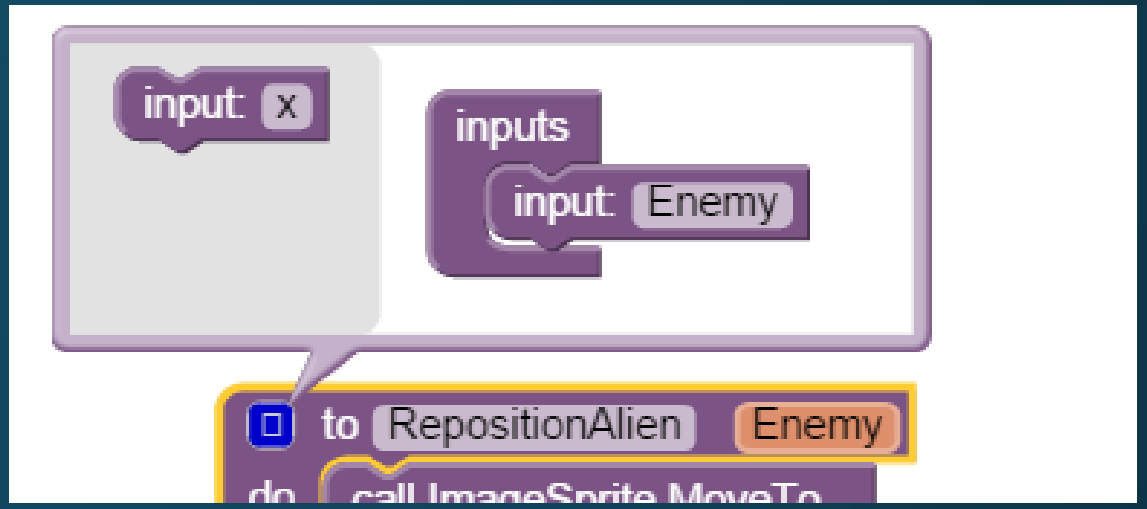


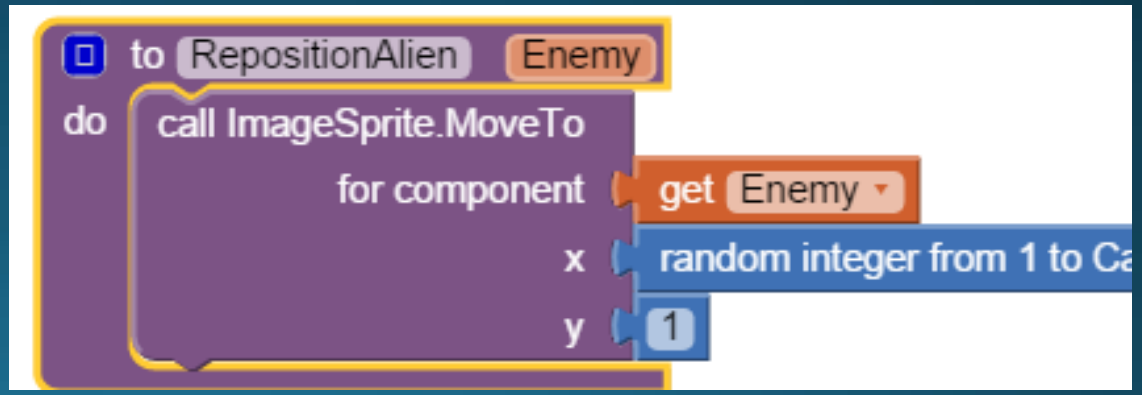



# Let's make our function dynamic

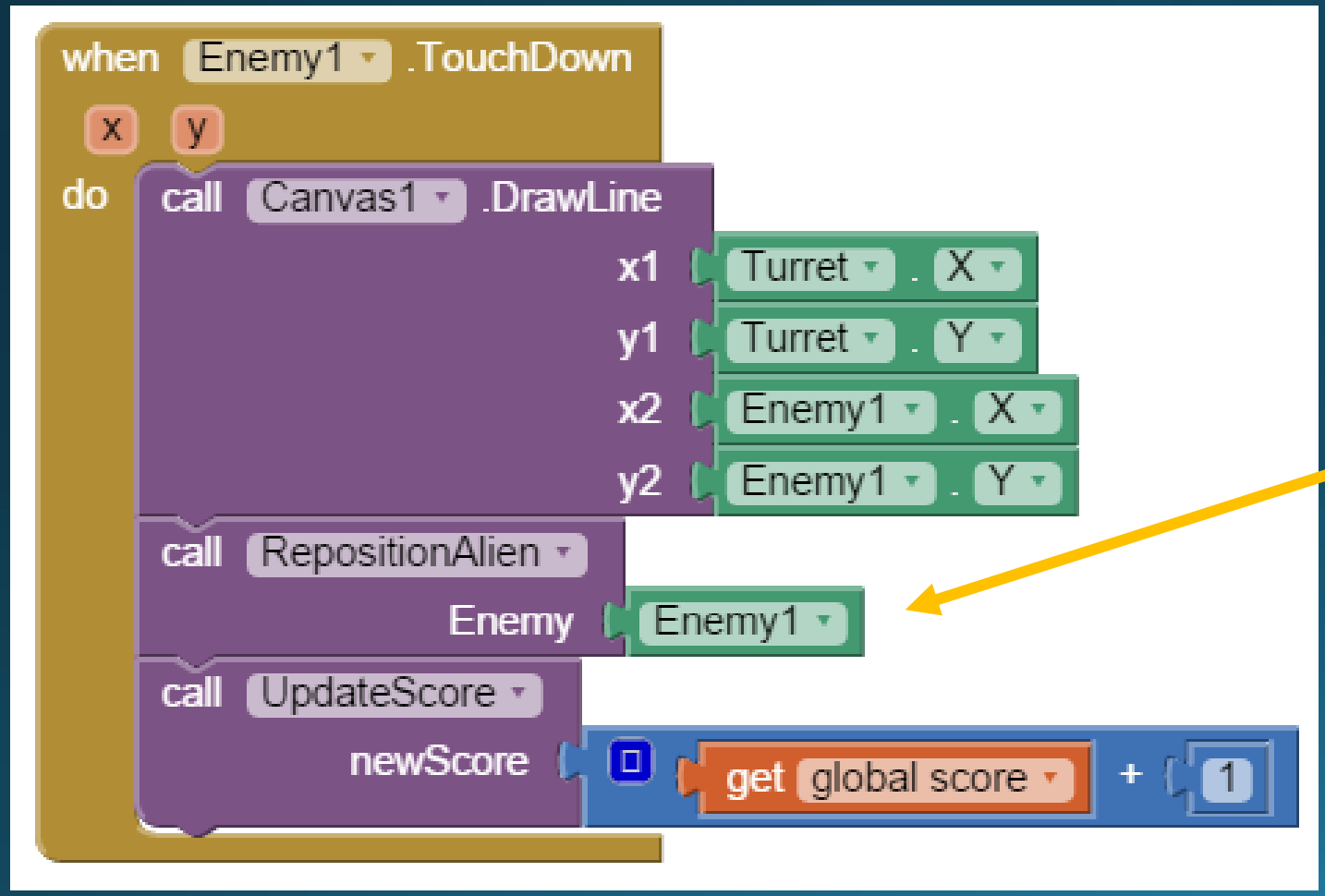
- Edit your RepositionAlien function to take one parameter named "Enemy"

- Replace the Enemy1.MoveTo block for a dynamic ImageSprite.MoveTo block.
  - Located under:
    - Any Component
    - Any ImageSprite
    - Imagesprite.MoveTo



# Result: (for enemy1)



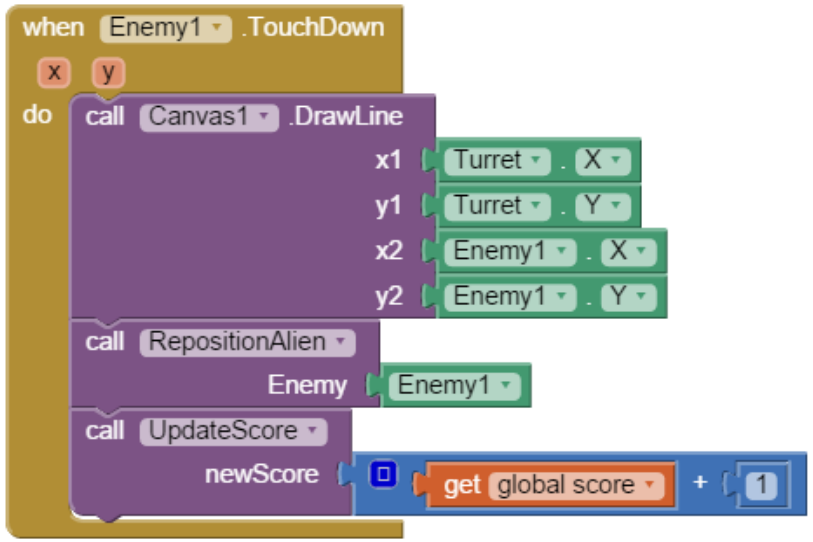

- Notice how your RepositionAlien block has an enemy slot

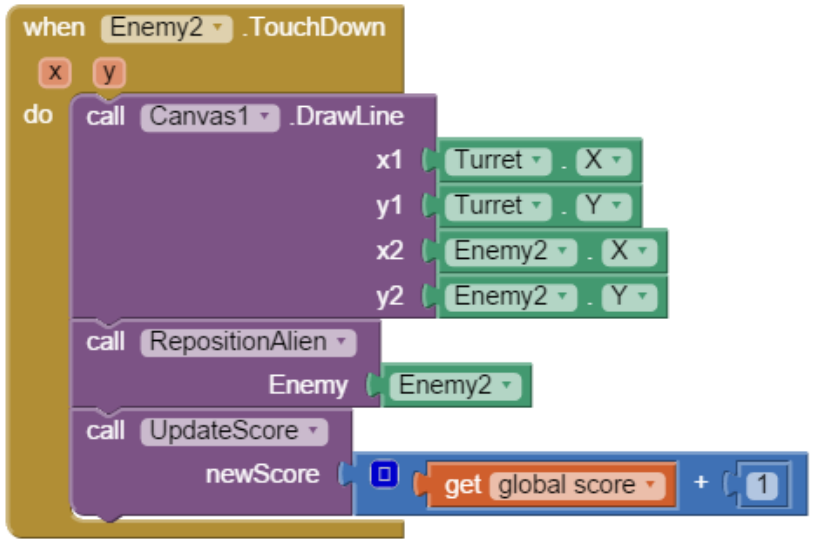

- To fill the slot drag in the

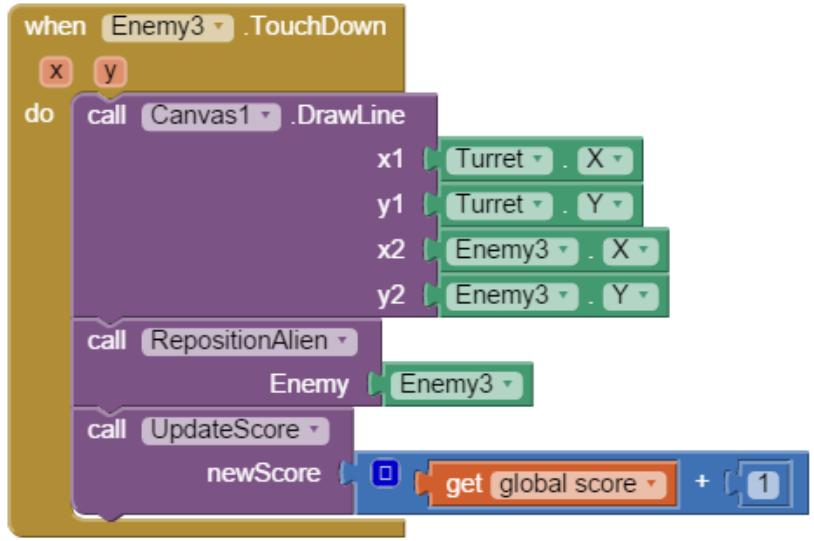

from the Enemy1 menu

# Enemy 2 and 3

- We could copy the code….



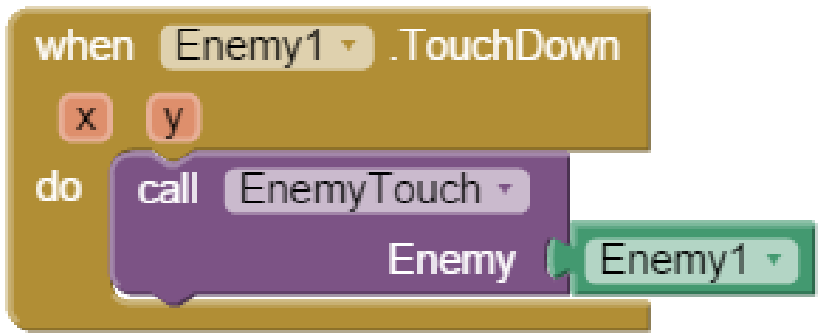

- But let's use a function instead.

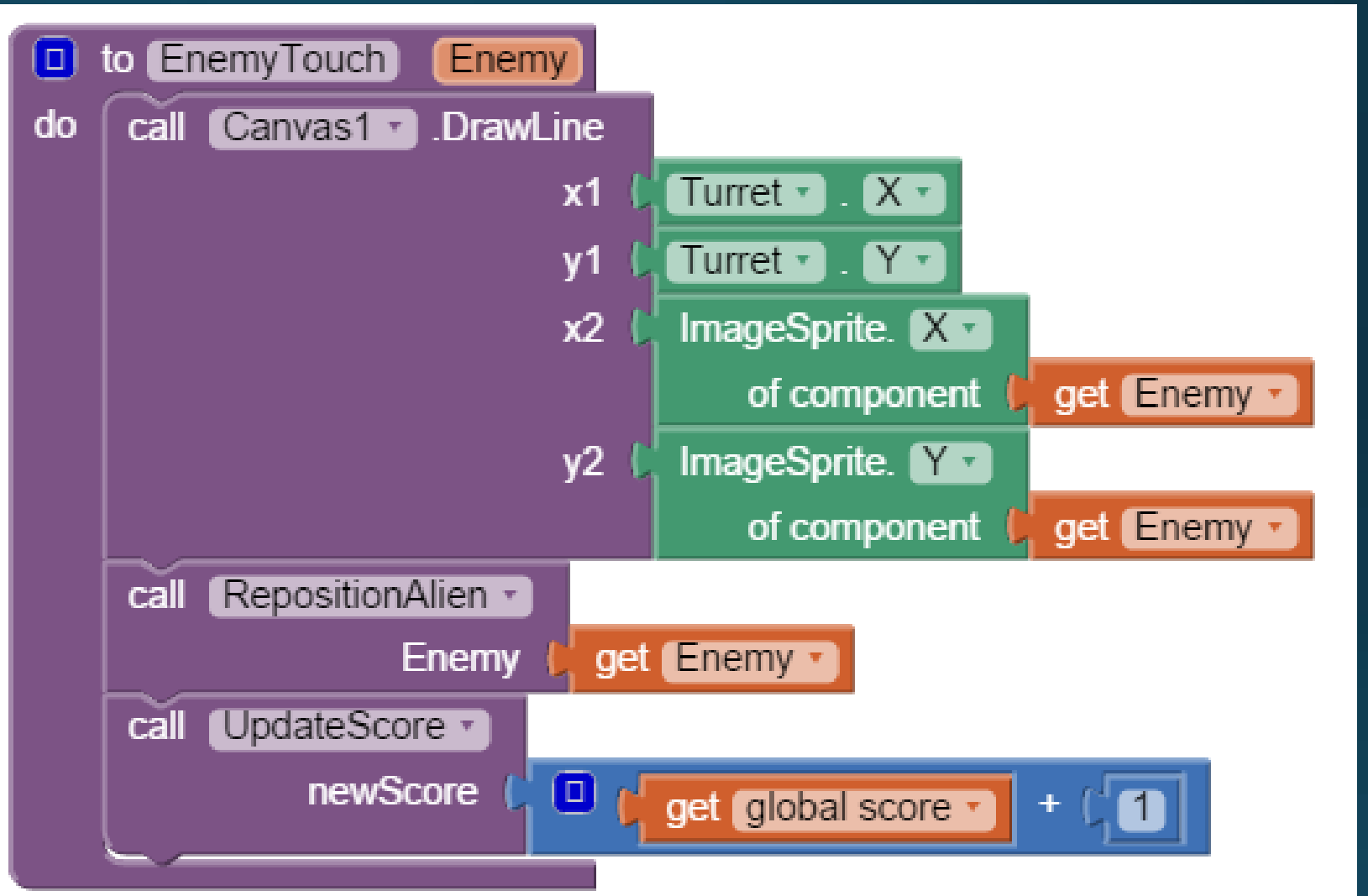
# Making a function:

- Drag in a new function definition named EnemyTouch and give it a parameter Enemy

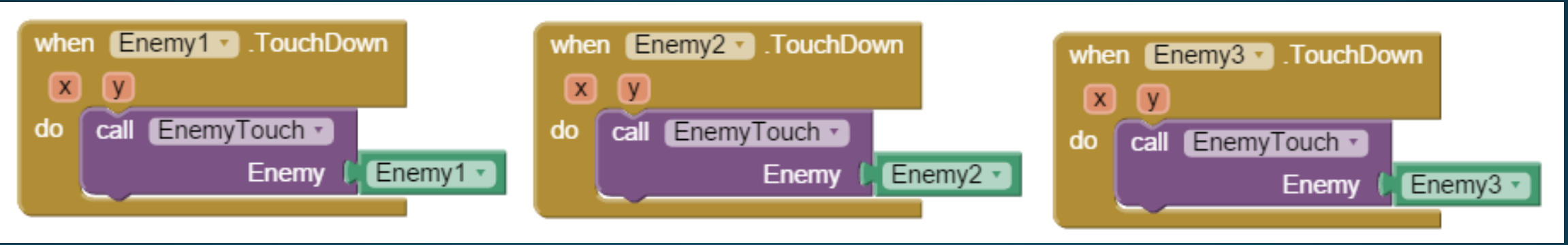

- Drag all the logic from the Enemy1.TouchDown event into this new function.
- There are three blocks within that say Enemy1 inside them.
  - Replace them with Dynamic Blocks.

# Result

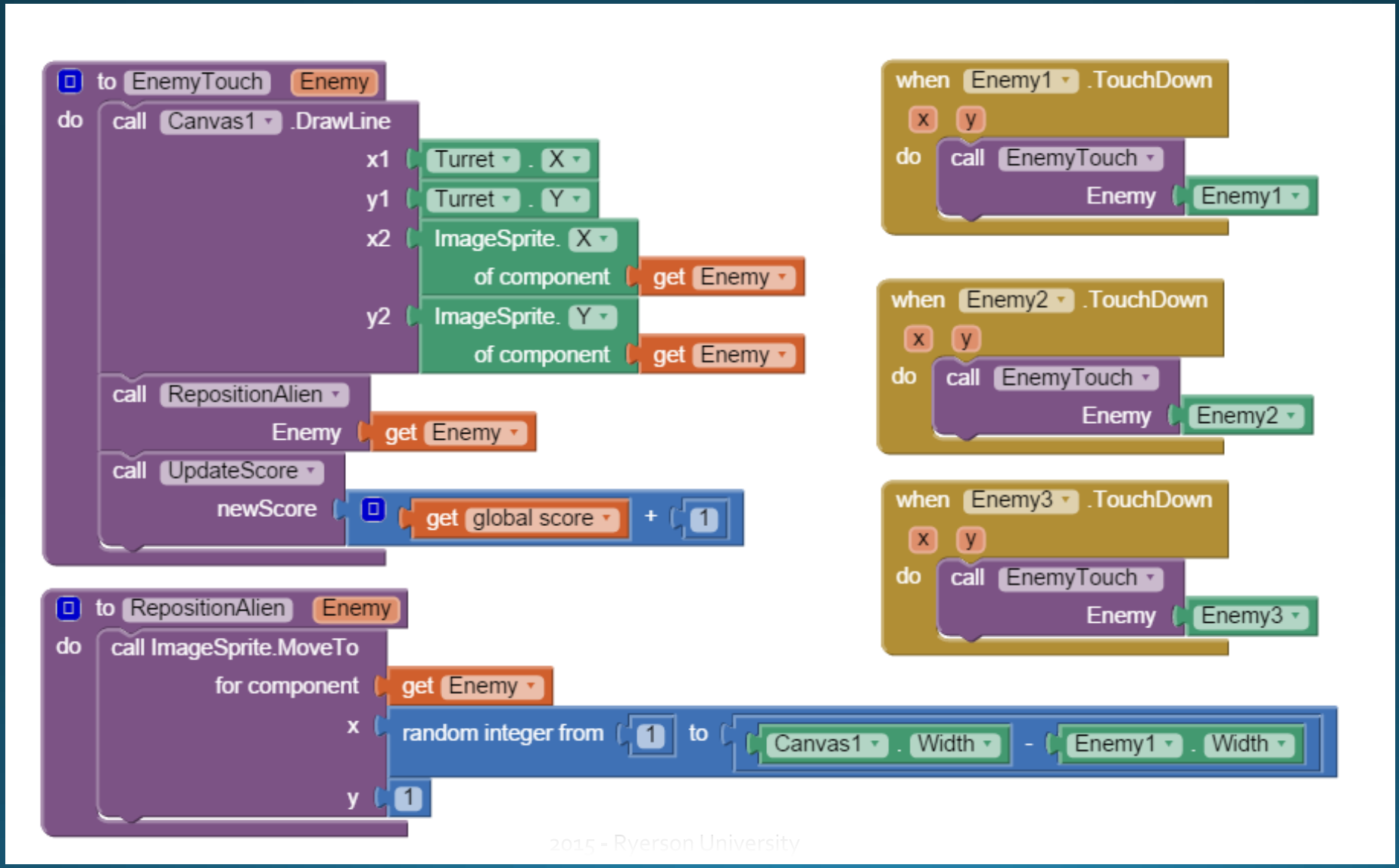

when Enemy1 .TouchDown
x y
do call EnemyTouch
Enemy Enemy1


to EnemyTouch Enemy
do call Canvas1 .DrawLine
x1 Turret . X
y1 Turret . Y
x2 ImageSprite. X
of component get Enemy
y2 ImageSprite. Y
of component get Enemy
call RepositionAlien
Enemy get Enemy
call UpdateScore
newScore get global score + 1

# Now… we can copy/paste.

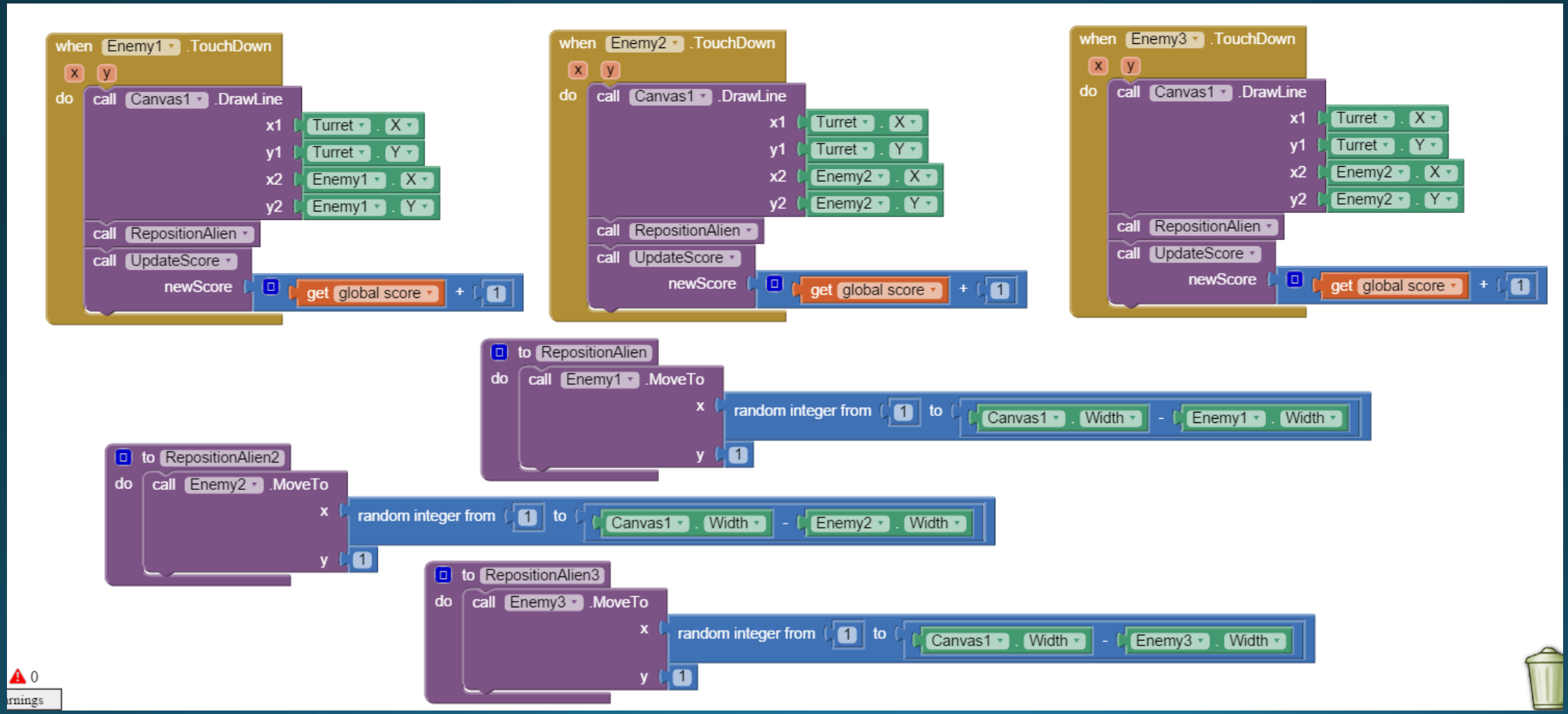


# Result:




2015 - Ryerson University

# Compared to:


when Enemy1 .TouchDown
x y
do call Canvas1 .DrawLine
x1 Turret . X
y1 Turret . Y
x2 Enemy1 . X
y2 Enemy1 . Y
call RepositionAlien
call UpdateScore
newScore get global score + 1
when Enemy2 .TouchDown
x y
do call Canvas1 .DrawLine
x1 Turret . X
y1 Turret . Y
x2 Enemy2 . X
y2 Enemy2 . Y
call RepositionAlien
call UpdateScore
newScore get global score + 1
when Enemy3 .TouchDown
x y
do call Canvas1 .DrawLine
x1 Turret . X
y1 Turret . Y
x2 Enemy2 . X
y2 Enemy2 . Y
call RepositionAlien
call UpdateScore
newScore get global score + 1
to RepositionAlien
do call Enemy1 .MoveTo
x random integer from 1 to Canvas1 . Width - Enemy1 . Width
y 1
to RepositionAlien2
do call Enemy2 .MoveTo
x random integer from 1 to Canvas1 . Width - Enemy2 . Width
y 1
to RepositionAlien3
do call Enemy3 .MoveTo
x random integer from 1 to Canvas1 . Width - Enemy3 . Width
y 1
0
rnings

# Loose ends:

- If we test our app at this point, the basic touch to respawn works for all three enemies.

- For our challenges, we will look at implementing the other events and making the game more engaging.

Take a break

# Challenges Ahead.