**Functions Lab**

Functions are another key building block in computer programming. They allow you to teach your computer program how to do things.

Think of a function as a recipe or a set of instructions. Let’s imagine I want to make a function called makeCereal()

The instructions to do that would look something like this:

* Get a small bowl
* Get a spoon
* Poor 1-2 cups of cereal into the bowl
* Poor 1/2 cup of milk into the bowl

Once you’ve taught your computer how to do something via a function, the beautiful thing about it, is you can just tell your computer to do it again! (But this time you don’t have to repeat the instructions, your computer program now knows how to makeCereal). Another way to think about a function is designing a cookie cutter. Once you’ve made the cookie cutter, you can stamp as many cookies as you want much easier, and they will have whatever level of detail you put into the cookie cutter.

**A Simple Example!**

Here is a basic example of a function:

// This function will print out the words “Hello World” to the computers console.

**func** helloWorld() {

print("Hello World")

}

Lets break this down real quick.

* **func** - This is how you start a function.
* helloWorld - This is the name you are giving to your function. By calling this name later in your program (calling means, telling the computer to run the function) the computer will know to run the code in this function.
* () - This is where the parameters for the function are defined. These are the inputs variables the function can use. In this case, we have no input variables defined. If we had defined a variable it could have looked like this(myVariableName: Int)
* { ... } – These are called “curly braces”. The first curly brace { tells the computer where this functions body starts. The last curly brace } tells the computer where this function ends. In between those curly braces is called the function “body”. This is where you actually put your instructions.
* print("Hello World") ... – This is our functions body. For this example, we did something super simple. We just printed out the words “Hello World”. We could have put as many lines of code in here as we wanted to perform our function
* // This ... – This is a comment. Comments are things written in your computer program that your computer will ignore. Think of it as a way to leave notes in your program. Whenever the computer sees the symbol // it will ignore everything else that comes after that line. Comments are important to help write down descriptions of what different code does.

One last note on functions. In code there are two ways functions are represented. You can “CALL” the function, and you can “DEFINE” the function.

In the figure below on line 75 we are calling the function “setupEnemyMovement”. When we call the function, it runs the code inside of that functions definition. Think of this as stamping the cookie cutter.

![A screenshot of a cell phone

Description automatically generated]()

In the next image on line 80 we are defining the function “setupEnemyMovement”. Thing of this as changing the cookie cutter so its stamp is different.



**Hands On!**

Time to get our hands dirty. Our game has lots of functions. Lets try running a few different ones and see how it changes our game.

Inside the Enemy.swift file, you will find a function called “setupEnemyMovement()”. Go to this functions definition. Once you find it you will see that it has several statements all, which are comments.

If you look at the comments, they are actually code that we are hiding from the program. We can remove the comment markers, to make the code actually run. Remove the // from the moveEnemyToPoint statement. And add // before every other statement in this function body. Your code should look like this when you are finished. With only the line 82 uncommented:

![A screenshot of a cell phone

Description automatically generated]()

What happens? If you look at the code for moveEnemyToPoint() Why do you think the enemies move the way they do?

There are three more advanced movement functions already built for your enemies. Try each one by uncommenting it (uncommenting, means removing those comment marks) Make sure you only uncomment one at a time. (Add comment marks // back to the other statements)

* moveEnemyBackAndForthForever 20 pts \_\_\_\_\_\_\_\_\_
* moveAndWaitEnemyForever 20 pts \_\_\_\_\_\_\_\_\_
* moveEnemyRandomly 20 pts \_\_\_\_\_\_\_\_\_

Once you’ve taken time to try each one. Choose your favorite, and keep it as the movement behavior for your enemies in this game.

(Teachers Note: You may notice that our code for these movement functions uses SKAction over and over. SKAction is a library of functions pre-built into swift that Apple built for us! Under the covers these are just more functions, with more instructions on how to do things at a lower level. By using these pre-built functions we are able to “abstract” away the details we don’t need to worry about, like how to actually move a sprite.)

***BONUS EXCERCISE: (Do this last)***

*Its not too hard to write your own logic to move! The movement functions already written use Swift’s built in “Actions” library. Examples of “moveTo”, “wait”, and some boiler plate to repeat and run these actions forever already exist in the pre-built functions. Feel free to copy + paste, modify and make your own custom movement logic inside of the moveEnemyCustom function I’ve started for you. Can you make your enemies fly up and down instead of back and forth? How about move in a square? Zig-zags? Have fun messing around!*

* What did you make your enemies do different? 30 pts \_\_\_\_\_\_\_\_

All the code we work on today is available at <https://github.com/Unome5548/Digigirlz2020> If you want to keep working on your game, you can download it there. Email me at [mipatte@microsoft.com](mailto:mipatte@microsoft.com) if you have any questions or need help!