Dice Game

Here we will talk about loops in our code.

Loops

We've already seen functions in our code and how they hold blocks of code that we can reuse.
Functions aren't always the best fit for our problems though. Lots of the time a loop better suits our needs. Let's go over the two main types of loops we will use.

For Loops

The "For Loop" is used to repeat through a block of code a certain amount of times.
As usual let's get an example up and then walk through it.

```
for(int i = 0; i < 5; i++) {
        Debug.Log("Hello World");
}
```

The output for this would be :

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

As we said, the for loop will execute a block of code a certain amount of times. In our case printing "Hello World" five times.
Let's go over the three main parts of making a `for` loop which you can remember with the mnemonic F.O.R. conveniently.

- **F**irst We need to initialize a variable that keeps track of how many times we've looped. Typically you'll see `int i = 0;` . This is common because whole numbers (`int`) nicely keep track of the number of loops as we don't need to worry about decimals. `i` is typically used as it can stand for "iterator" though you can use anything. `0` is the start value as computers actually start counting at 0, again you can set it to whatever and sometimes you will want to.
  This is value is only set once.

- **O**perator Than we need an operator to figure out if we should continue looping.
  The second part of the statement, in this case `i < 5;` , will determine, via a boolean or boolean statement, whether we continue looping. This can be whatever you want again as long as it gives you a `true` or `false` value.
  This is done every iteration of the loop before the code is executed.

- **R**epeat Lastly, if we do another loop we do something to our variable to progress the loop.
  The last bit of the statement in our case is `i++` which simply increases `i` by one. You can do whatever you here not just `++` (that feels like a trend).
  This is done after the code is executed.

Now let's show a more visual way to think of a loop.
You can imagine :

```
for(int i = 0; i < 5; i++) {
        Debug.Log("Hello World");
}
```

turning into :

```
int i = 0;
if(i < 5) {
        Debug.Log("Hello World");
}
i++;
// Loop back to the if!
```

This misses some nuances of the loop but will do for our purposes.
Use loops to repeat code a certain amount of times or until a condition is met.

While Loops

A "While Loop" is really the same thing as the `for` loop except with a different declaration.
A `while` loop only takes a boolean or boolean expression. For instance :

```
int i = 5;
while(i > 0) {
        Debug.Log(i);
        i--;
}
```

The output of this is :

```
5
4
3
2
1
```

This one is counting down for variety, no reason it couldn't do the same as the `for` loop. Since there is not too much different between `while` and `for` loops we are going to move right on to the next important part of loops.

Infinite Loops

Infinite loops are dangerous. One wrongly placed `>` can quickly stop all of your code execution.
These infinite loops are caused when the boolean statement given to the loop will never become false. Our code will break because the computer will loop forever and never be able to break out of the loop. Eventually the computer will run out of memory and decide to exit your program with an error.
Here are two examples :

```
while(true) {}
// This should be fairly self explanatory.

int myInt = 5;
while(myInt < 0) {
        myInt++;
}
// myInt is being incremented NOT decremented and will never be less than 0.
```

Go ahead and try running these. Don't worry your computer won't burst into flames, it will just tell you that something is wrong.
When you get errors like this try stepping your code carefully. Often there is a single character out of place. Other times you may need to look more at the overlying logic of your code. Perhaps an `if` statement inside the loop has a loophole that will make it be never true.

Loop Statement Keywords

There are two main statements used to control our loops, "break" and "continue".
These aren't too complex, can be very useful, and do exactly what they say.

Break

Break simply breaks out a loop. It is written simply as `break;` but it must be within a `for` or `while` loop.
When the break is executed the loop is stopped where it is and program moves to after the loop.

```
for(int i = 1; i < 10; i++) {

        if(i == 5) {
                break;
        }

        Debug.Log(i);
```

```
}

Debug.Log("The loop is over.");
```

The output of this will be :

```
1
2
3
4
The loop is over.
```

As you can see once `i == 5` we "break" out of the loop and non of the next iterations are done.

Continue

Continue simply moves onto the next iteration of a loop. It is written as `continue;` but is must be within a `for` or `while` loop.
`continue` does not stop the whole loop it only skips the rest of that iteration.

```
for(int i = 1; i < 10; i++) {

        if(i == 5) {
                continue;
        }

        Debug.Log(i);

}

Debug.Log("The loop is over.");
```

The output of this will be :

```
1
2
3
4
6
7
8
```

```
9
The loop is over.
```

You'll notice that `5` is missing from the output as when `i == 5` we "continued" onto the next iteration.

On Your Own

Loops can seem simpler on the outside than perhaps the boolean logic we just went through, but they can become very versatile.
Here's a few ideas to practice with.

1. Make a loop that prints only even numbers, then make it print only odd numbers.
2. Try putting a loop inside another loop and figure out how it prints out some numbers.
3. Print out a mathematical sequence of numbers. Exponential, the Fibonacci sequence, multiples of x, or factorials perhaps.