Interactive Role Play - Commands

Introduction

This role playing activity aims to familiarise children with the ways in which we **use code to *talk to computers***. It also introduces the practices of **testing** and **debugging** code collaboratively.

In this activity one participant pretends to be the computer, executing a sequence of commands (the `script`), which other participants have written.

By tapping into the children's well-established knowledge of their own bodies and how to move around, as well as their ability to write on paper and talk to each other, we try to make basic programming concepts and practices easy to *assimilate*, and hopefully fun too!

This activity may be played before children start to use Scratch, possibly on the very first Code Club session.

Concepts covered

This role playing activity should help children familiarise with:

1. the idea of coding as *giving instructions*
2. the difference between giving instructions to another person, and instructing a computer
3. the concept of `command` as the building block of coding, as a simple, self-contained instruction
4. the concept of `script` as a sequence/cluster of `commands`. This term will be used in Scratch so try and stick to it (instead of using *program* or other synonyms)
5. the practice of `testing` and `debugging` code, which should hopefully **change the way kids look at errors**: not as a *bad mistake*, something to erase, but as an opportunity to understand how the *computer thinks*.

Ingredients

- post-its (schools may not have these)
- pencils or sharpies (usually schools have these)
- large paper sheets (A3 or so, where kids arrange the code blocks/post-its)

Method

Who's done coding before?

Before playing the activity, you may want to **get children thinking about how we interact with computers**, and **what coding means to them**. You may start with something along these lines

> Have you done any coding before?

They may say *no*, but it's not unlikely that some kids will surprise you :)

You can then ask them if they have ever **explained** to someone how to get to their house, or **described** to a friend how to find the toilet.

You should get more *yes* answers this time.

Point out that these actions are *coding*.

> Coding is **instructions**. You write the instructions, and someone or something else follows them.

Talking to computers

At this point you may want kids to think about the differences between giving instructions to another person, and giving instructions to a computer.

> Computers are very powerful and can do many things, but if you don't tell them what to do, they'll just sit there and get dusty. Unlike you and me they don't get hungry, so they won't get up and look for food. They won't get lonely, or bored, or sleepy. They just sit there and wait for you to tell them what to do. In

> this Code Club, **we will learn how to talk to them**.

Let’s pretend to be a computer

> In order to learn how to talk to computers, we can play a game where one of us pretends to be a computer, and the rest of us has to code her or him.

In this game you can only use 4 `commands`:

1. step
2. right
3. left
4. grab

You can use these commands as many times as you want, and in any order.

Post-its with individual commands written on

The volunteer plays the computer

Do a first run where you pretend to be the computer so that kids understand the commands and how the activity works.

> We’ll start with me playing the computer, and **you will code me**!

> Let’s try. I am here and you want me to grab that box (or whatever) there.

> Think about the order of commands you need to give me to get there and grab that thing. What command should I start with? And then?

Ask one child at a time to give you one command, and then execute it.

Make sure to not execute commands that your fictional interpreter doesn’t understand. For instance, if a child says *move* stay still for a few seconds, then ask them why you didn’t move. Point out that *move* is not a command our fictional computer understands, and ask them which command we should use instead.

> When we give computers commands, we need to be very **precise**. They’re not like people, so if they don’t understand what you want, they just don’t do anything or start complaining!

Children play the computer

After children have coded you to grab your target object, they can start coding themselves :)

Ask them to **form groups** of 2/3 people, and then hand out post-its, pencils/markers and a big sheet of paper, so that they can write down their code.

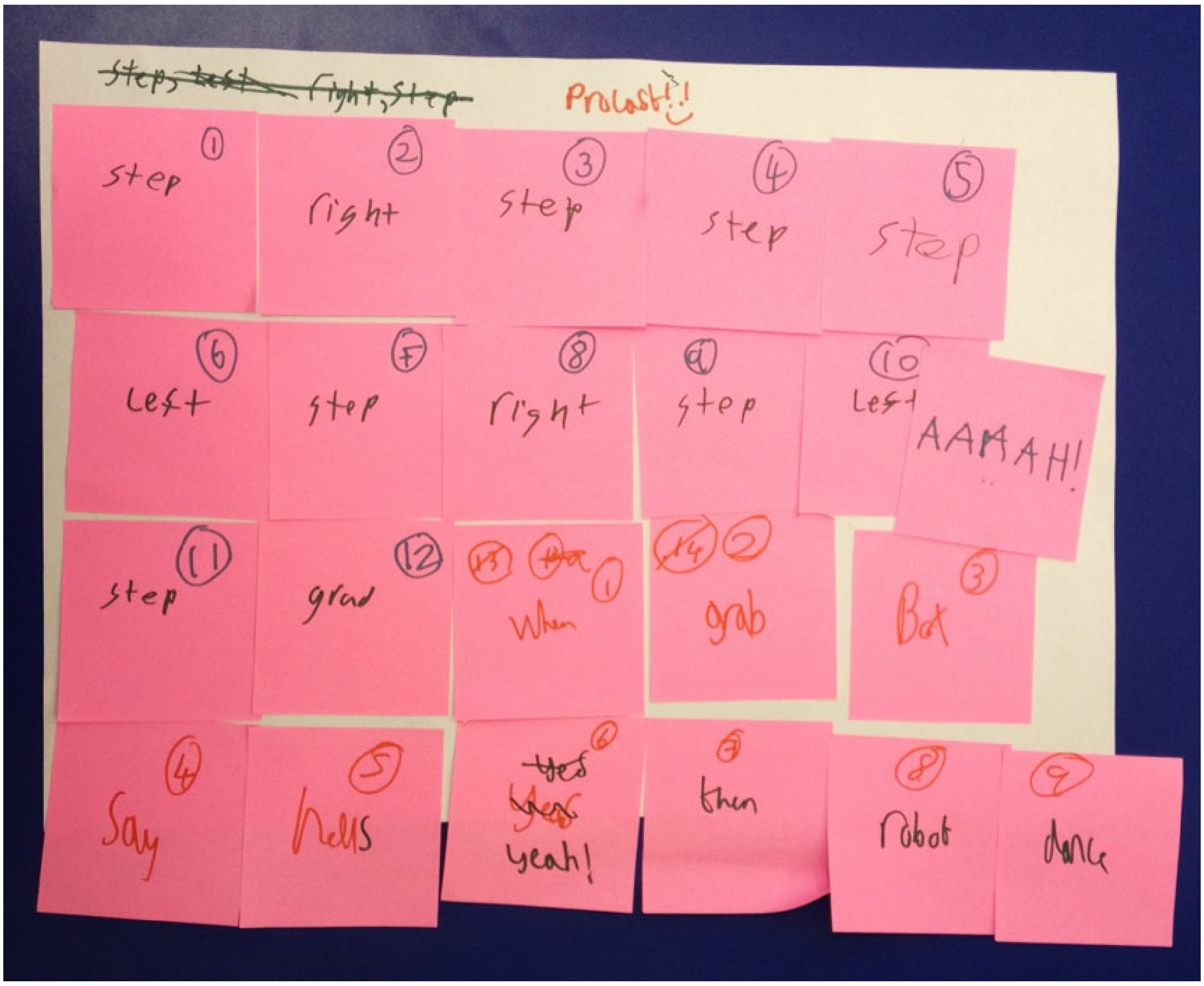
Let children decide what their target will be. It may be useful though, to have a shared target, so that later you can compare the different code groups have written.

Children will then start writing commands and sticking them on the sheet. They will probably start testing their `scripts` without you telling them to do so.

Let them code/test/debug for some 5 minutes, then ask one group at a time to demo their `scripts` in front of the Club.

This is a chance to introduce them to the practice of `testing`, and reassure them that it doesn't matter if they didn't get it right the first time: coding is about writing and testing and writing and testing again and again, moving code blocks around..

When children are demonstrating their scripts, you can also pause the execution of the script to point out that there is a `bug`, for instance if a script were leading a child into a table, or another obstacle.



Post-its sequence froms a script

Wrap up

> So what you have written down and stuck on your sheets are instructions for computers, which you can also call **scripts**.

> Real computers also work with scripts! You'll see that when we start using Scratch.

> Just like with post-its, you can **move code blocks around** and your script changes.

At this point, you could continue with the introduction to Scratch, or play more game(s) to illustrate other basic concepts, such as `events` (*when* this happens, do that), `variables` and `conditionals`.