FEATURE

GP, SNAP!, SCRATCH
NEW STRATEGIES FOR NEW CONCEPTS

What's different with block-based programming?

STORY BY Sven Jatzlau and Ralf Romeike

Choosing the right programming language for your class is a tricky thing. The designers of the language have made choices about what it looks like and how easy it is to use. The choices made in block-based programming languages such as Scratch, Snap!, or GP make them very different from text-based languages like Java or Python. In these languages, we can find new concepts, and with them new approaches to solving problems. This also means we need to make learners aware of these new concepts and possibilities.

What's different?

In 2005, Scratch made its debut. Developed by the Lifelong Kindergarten Group at MIT, it represented a new approach to introducing programming to learners: based on the ideas of 'low floor, wide walls, high ceilings' by Seymour Papert, its popularity grew rapidly. At the time of writing, some 22 million projects, guides, animations, and games have been shared by a continuously growing user base of 19 million users. In schools, block-based languages display a similar success: compared to text programs, block-based environments enable students to achieve better results on average. They achieve these results faster, and with more motivation. At the same time, however, these block-based visual programs also introduced new concepts such as nesting of sprites, events, or first-class objects. While several of these concepts are found in Scratch, a number of concepts outlined in this article are only found in its sibling languages, Snap and GP. As they didn't undergo many of the simplifications found in Scratch, these languages lend themselves to secondary education. Teachers need to know something about the different concepts in order to more effectively teach them to their students!

Instead of instantly executing the loop, making it hard to see what happened, the sprite will slowly rotate 360°

Delayed execution

Most of the new concepts exist for a pedagogical reason: making the language intuitive and easy to use. One of the ways this is reflected is that code scripts are executed slower than the environment (and the computer!) would normally allow. Without this delay, sprites could shoot out of the stage boundaries instantly, without the user being able to understand what happened, because they couldn't see. To protect users from experiencing this, all the loops, wait blocks and motion blocks were equipped with a delay. Snap! in particular takes this one step further: activating 'visible stepping' enables the user to not only see which block is currently being executed, but to control the execution speed manually. For debugging purposes, this is an invaluable tool!

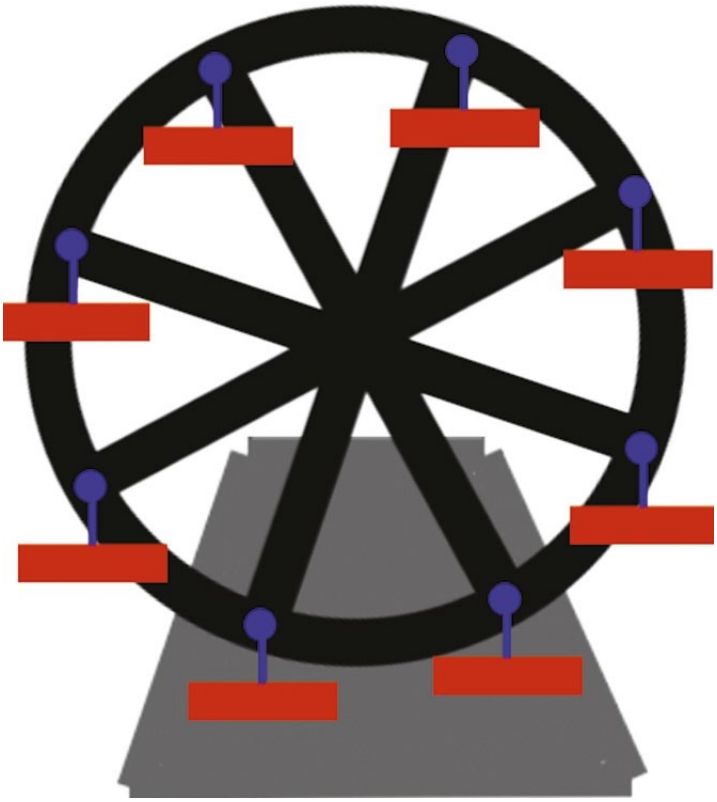
Nesting

Visual programming creates new approaches to solve new problems. Sprite nesting is a great example of this. It creates a hierarchy between two objects/sprites by making one the owner (or 'anchor' in Snap!), and the other a part. While Scratch doesn't support the nesting of objects, Snap! and GP do. Dragging and dropping one sprite from the sprite palette onto the other one on the stage nests the two objects into each other. From that point onward, they'll move and turn as a single unit, reference each other using the anchor/ owner and

parts blocks, but will still be considered two objects for the purpose of the program. This also means that they can still rotate themselves – it's easy to think of it as a part-whole structure, like a car: it consists of many single parts, like tyres, mirrors, the engine, etc. When the car drives and turns, all of its parts (naturally) turn and move with it. However, parts can be removed and added, and some can even turn by themselves (like the tyres) without it affecting the rotation of the entire car. In block-based programming, this concept enables the user to detect which part of an object collides with another object, to create composite objects consisting of smaller parts, and even to create physics-based simulations!

First-class objects

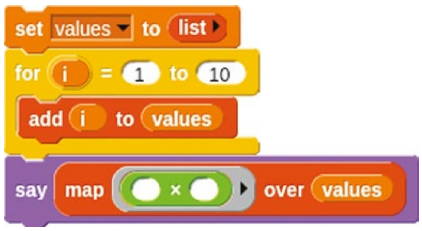
Whilst there are loads of complex definitions of the term, essentially the idea behind the concept is that it should be possible for every object in a programming environment to be used freely – there should be no limitation on the context in which an object can be placed: as the value of a variable, an input to a procedure, etc. An example of this can be seen in Java: Integers, Strings and Booleans can be used as the input, or parameter, of a method, or as the return type. They can also be assigned as the value of a variable. It seems that there are no limitations on what we can do with these data types; this makes them first-class objects. But what about methods? Methods can't be assigned to a variable, used as the input to another method, or returned by another method. Do these limitations make them second-class objects?



In this Snap! program, when the ferris wheel turns, the cabins turn with it. Each individual cabin constantly points downwards to simulate real physics

> BLOCK-BASED ENVIRONMENTS ENABLE STUDENTS TO ACHIEVE BETTER RESULTS ON AVERAGE

In Snap! and GP, all objects are first class, therefore putting no limits on what the users can 'do' with them. This concept is closely related to the term 'higher-order functions': being able to store methods (or blocks in our case) in lists allows us to use them as inputs for other functions, therefore making them higher-order functions!



Mapping the function over values returns a list of 1, 4, 9, and so on. Higher-order functions made simple!

Event-based programming

There have always been event-driven programming languages (events such as user input, sensor readings, or messages from other threads control the flow of the program). This is the type of programming that can be found in Scratch, Snap!, and GP. Because events are intuitively understood as something one can react to, learners instinctively program parallel solutions to problems: an object reacts to a button input with one script, and to touching another object with another script. If both events occur at the same time, the object does both things at the same time! Learners implement these solution without even realising why that's a hard thing to do in other languages. This opens up the possibility of learning about concurrency and dependency in simple, intuitive environments and contexts.

What does this mean?

The growing presence of these programming languages brings great, new possibilities to classrooms. At the same time, however, new teaching strategies are needed in order to teach these new concepts and their possible applications. The more concepts learners understand, the more options they have to choose from to implement their solution, and more complex problems may be solved. (HW)