The WBEngine FishTank

Here we introduce a totally different way of using the WeeBee engine. In Story-Writing-Coding, character actions were synchronised by making each command, like **pip.jump();** last for a certain length of time; the default was 2 seconds. But there's another way of coding where we do not enforce this constraint. Instead we give the actors a speed, and use this to make them move.

This approach is very useful for KS2, where children must "use sequence, selection, and repetition in programs; work with variables and various forms of input and output". In fact, programming this way is easier than Story-Writing-Coding. From a cross-curriculum point of view, this approach links nicely with mathematics and physics.

We do need to understand what an object's *motion* really is. That's simple, if something has a *speed* or a *velocity* then it is moving. The larger its velocity, the larger the distance it covers in any interval of time. It moves faster! The code for the FishTank has a new function called "swim" like this, **pip.swim(speedX);** so if you wrote **pip.swim(10);** then pip would swim with a speed of 10.

Perhaps the best way to understand the FishTank is to look at some code. In the code shown below, we make the fish move to the left then the right bouncing between two obstacles. The code is **aFish1.cde**

Line	Statement	Comment
7	float x;	Here we *declare* the variables we shall use in the rest of the program. [The effect of this is to reserve space in memory for the variables]
8	float velyX;	
9		
10	public void once() {	Start of the WeeBee function "once" which runs one time when play is pressed
11	showGrid();	Tells the engine to show the Cartesian grid
12	velyX = 10;	Sets the value of *velyX* to 10. This is the starting value of the fish velocity in the x-direction
13	}	Curly bracket shows the end of the function "once"
14		
15	public void loop() {	Start of the WeeBee function "loop" which runs continually when once is done
16		
17	add(pip,10,0);	Here we add characters and scenery as usual. Pip plays a special role. She does not move, but she determines how long the loop code will run.
18	add(puffy,30,20);	
19	add(tree,20,20);	
20	add(tree,50,20);	
21		
22	pip.rest(20);	This tells Pip to make the loop run for 20 seconds
23		
24	x = puffy.getX();	This gets the current position of the fish, which we use in lines 26 and 29
25		
26	if(x > 50)	Here we test if the fish has reached the right boundary (at 50) and if it has …
27	velyX = -10;	… then we set the x-velocity negative (so that the fish moves to the left)
28		
29	if(x < 20)	Here we test if the fish has reached the left boundary (at 20) and if it has …
30	velyX = 10;	… then we set the x-velocity positive (so that the fish moves to the right)
31		
32	puffy.swim(velyX);	This line actually makes the fish swim with the velocity set by the previous code
33		
34	}	Curly bracket shows the end of the function loop()

The important parts of the structure of this code have been highlighted. Blue shows the start and end of each block of code associated with the functions **once()** and **loop().** The code itself shows the start of each block by a **{** and the end by a **}**. Declaring the variables used is shown as orange and declarations are usually made "up top". The "guts" of the program (which make the fish behave) are shown in yellow. Here, it's best to think backwards; line 32 makes the fish move, using its velocity, **velyX**. So how is this velocity given a value? Well, the lines before that set the value of **velyX**, i.e. lines 27 and 30. To understand which line is used, we need to look at the *selection* statements, lines 26 and 29. They use the value of the fish's location **x**, and this is grabbed at line 24. So if the fish moves to the right, and past the right see-weed, its velocity is set to -10, and if it moves to the left and strays past the left see-weed, it will start moving to the right. Remember that the lines in the **loop()** function (17- 32) are run for 20 seconds, as specified in line 22, so the fish will effectively "bounce" between the see-weed.

So how does the above code relate to the KS2 Computing programme of study? Well, lines 26 and 27, and lines 29 and 30 refer to *selection*, the **loop()** function refers to *iteration*, and lines 7,8,12,24,26,29,30,32 refer to *variables.*

Reflecting on this code, there are two key variables which are used, the x-location of the fish (**x**) and the velocity of the fish **velyX**. This could be simplified, to remove the *explicit* mention of **velyX**. We could use a function to define this. This helps, since the code could easily be extended to include more fish, or even jellyfish. Here we are moving into the territory of OOP. So let's revisit the above code using this new model.

Line	**Statement**	**Comment**
7	float x;	Here we *declare* the variables we shall use in the rest of the program.
8		
9	public void once() {	Start of the WeeBee function "once" which runs one time when play is pressed
10	showGrid();	Tells the engine to show the Cartesian grid
11	add(puffy,30,20);	Add the fish
12	puffy.setVelyX(10);	Set the initial x-velocity of the fish
13	}	Curly bracket shows the end of the function "once"
14		
15	public void loop() {	Start of the WeeBee function "loop" which runs continually when once is done
16		
17	add(tree,20,20);	Here we add characters and scenery as usual. Pip plays a special role. She does not move, but she determines how long the loop code will run.
18	add(tree,50,20);	
19	add(pip,10,0);	
20		
21	pip.rest(20);	This tells Pip to make the loop run for 20 seconds
22		
23	x = puffy.getX();	This gets the current position of the fish, which we use in lines 25 and 28
24		
25	if(x > 50)	Here we test if the fish has reached the right boundary (at 50) and if it has …
26	puffy.setVelyX(-10);	… then we set the x-velocity negative (so that the fish moves to the left)
27		
28	if(x < 20)	Here we test if the fish has reached the left boundary (at 20) and if it has …
29	puffy.setVelyX(10);	… then we set the x-velocity positive (so that the fish moves to the right)
30		
31	puffy.swim();	This line actually makes the fish swim with the velocity set by the previous code
33		
34	}	Curly bracket shows the end of the function loop()

WeeBee Fish-Tank API

Movement (Implicit)	
puffy.swim();	Moves with a constant speed set by a setVely function.
puffy.move();	Preferred method. Makes use of several objects easier
Movement (Explicit)	
puffy.swim(velyX);	Moves with constant speed passed as an argument.
puffy.swim(velyX,velyY);	Requires variables for each object. Not preferred method
puffy.move(velyX);	
puffy.move(velyX,velyY);	
Sets	
puffy.setVelyX(velyX);	
puffy.setVelyY(velyY);	
puffy.setVelyXY(velyX,velyY);	
puffy.setX(x);	
puffy.setY(y);	
Gets	
puffy.getVelyX();	
puffy.getVelyY();	
puffy.getX();	
puffy.getY();	
Immediate Image change	
puffy.looksLike(fname);	
puffy.lookslike(fname);	