WeeBee Code Reference (Schools Edition)

CBP 01-11-19

Methods

The table below lists the methods currently available for Actors and Props. The only difference between Actors and Props is that Actors can display emotions. The organization follows a linguistic classification, which stems from the engine's progeny in Story-Writing-Coding. May methods have several versions; e.g., the simplest **jump();** can be called without a parameter, but the jump height can be specified like this **jump(50);**

All methods (actions) take the same time. This is currently set to 2.0 seconds. This can be changed, as explained below. The term "dObj" (dynamic object) refers to either an Actor or a Prop.

Process: Material: Transformative: Enhancing: Motion "move-at"		
Basic Function	*+ param*	
flipH();		horizontal flip
flipV();		vertical flip
spin();	spin(speed);	spin at location
hover();	hover(speed);	same as spin
	spinV(speed);	rotate about vertical axis
jump();	jump(height);	jump
rest();		rest or pause

Process: Material: Creative:		
hide();		hide
show();		un-hide

Process: Material: Transformative: Elaborating: size		
	shrink(scale);	shrink by scale e.g. 0.8
	grow(scale);	grow by scale e.g. 1.2
	squishH(scale);	horizontal scale change
	squishV(scale);	vertical scale change
	squishHV(scaleH,scaleV)	scale change both horizontal and vertical

Process: Material: Transformative: Extending "possession"		
	pickup(dObj);	pick up a Prop or an Actor
	putdown(dObj);	put down a Prop or an Actor

Process: Material: Transformative: Enhancing: motion "move-to"		
	flyto(X,Y);	moves in a straight line to (X,Y)
	flyto(dObj);	moves in a straight line to a Prop or Actor
	walkto(X);	moves horizontally to X, in a straight line
	hopto(X);	makes a parabolic trajectory to X
	runto(X);	moves horizontally to X with a wobble
	flapto(X);	moves in a horizontal line to X with wings flapping (for winged creatures)
	stepto(X);	moves in a horizontal line to X with legs moving (for non-winged creatures)
	leapto(X,Y);	makes a parabolic trajectory to (X,Y)
	leaptoH(X,Y,H);	makes a parabolic trajectory to (X,Y), with control over the leap height
	flyto3D(X,Y)	like flyto(X,Y) but this variant gives a sense of perspective **EXPERIMENTAL**

Process: Mental: cognition		
	thinks(string);	string appears on canvas preceded by actor name
	thinks(string, fontSize);	
Process: Verbal: Projecting		
	says(string);	string appears on canvas preceded by Actor name
	says (string, fontSize);	
	says(string,fontSize,true);	string appears near top-right of actor/prop
	shouts(string);	string appears on canvas without the Actor name
	shouts(string, fontSize);	
	chirps(string)	plays a .wav file, the string is the filename, Cut off after anim time
	chirps(string, true)	same as chirps but plays sound at end of anim time.
	sings(string)	plays a .wav file, the string is the filename. Plays entire file
Process: Relational: Intensive Attributive		
	is(emotion); feels(emotion);	Only for Actors. Changes facial expression.
	appears(image);	Changes the image. Parameter is a string, or a proxy in Header.txt

Process: Existential:		
	add(dObj,X,Y);	adds Actor or Prop at (X,Y)
	add(scenery,X,Y);	adds item of scenery at (X,Y)
	add(scenery,X,Y,front);	*idem* but in front of Actors
Process: Mental: Perception		
	isNear(dObj);	returns boolean true if a dObj is close to another dObj
	canSee(dObj);	returns true if both Props or Actors are on the canvas

Scene Management		
	setScene(sceneID);	Select a built-in scene
	setScene("fname");	Load new scene from image supplied 900 x 600 .jpg
	changeScene("fname");	Load new scene from image supplied 900 x 600 .jpg and remove all previous scenery
showGrid();		Shows the grid on the canvas.
synch();		forces synchronisation of all Props and Actors to this point
tracePaths();		Breadcrumbs dropped when Actors and Props move – shows their paths

Turtle Graphics ("LOGO")		
	moveForward(dist);	Note. These can be accessed by WeeBees and Props.
	rotate(degrees);	
	setpenDown(true_false);	
	setpenDown(true_false, color);	

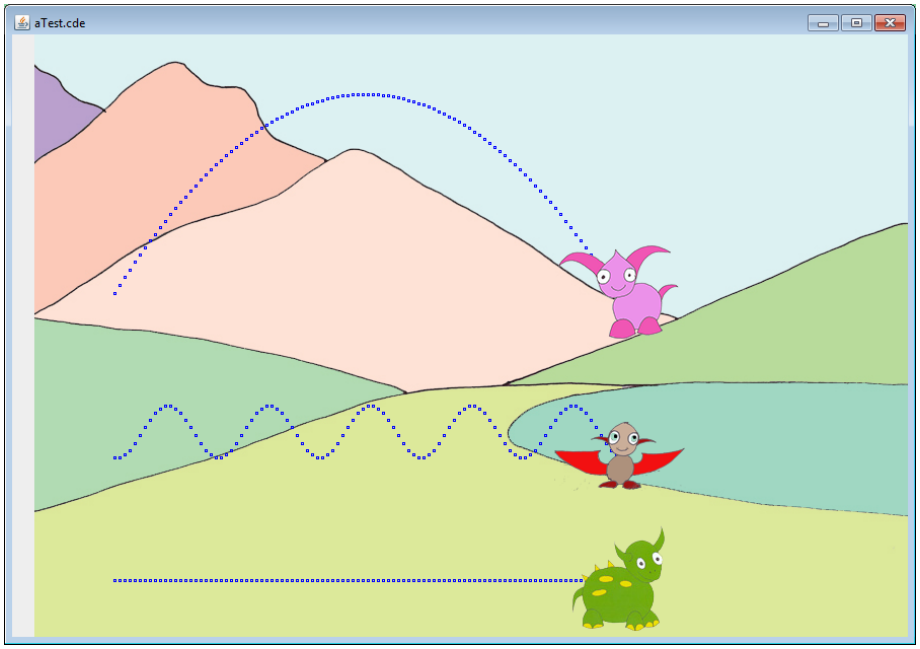
Non-Latent Methods. These return immediately, they do not wait 2 seconds. *Discussed in a forthcoming document.*		
	moveTo(X); moveTo(X,Y)	
	getX(); getY();	

	asksForNumFloat(string);
	asksForNumInt(string);
	asksForYesNo(string);
		User Input

	tells(str);	
	resize(scale)	
	becomes(emotion)	

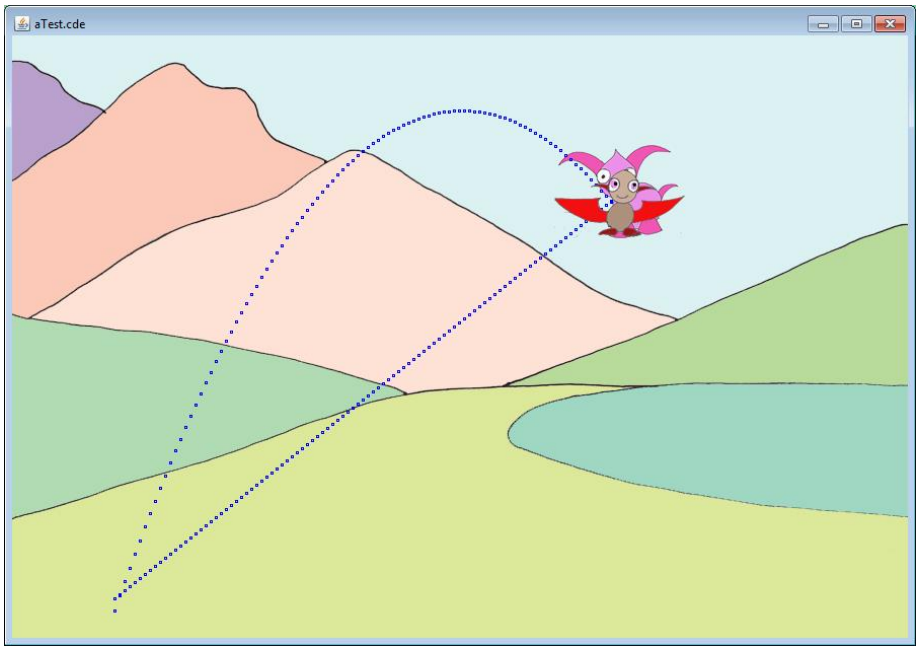
The Differences between the “Move-To” methods.

First, let’s have a look at the three methods that take a single parameter X. This means they are concerning with movement in the X-direction, so at the end of the movement there is no change in the Y-location of the Actor or Prop. These are shown in the picture below, where **tracePaths();** has been used to create the blue breadcrumbs.



Grog has used the **walkto(X);** method; he moves in a straight line. Flup has used the **runto(X);** method; she bounces up and down as she moves. Pip has used the **hopto(X);** method and her trajectory is the parabola expected of someone moving in gravity. The **flapto(X);** and **stepto(X);** methods behave like **walkto(X);** but either feet or wings move.

Now let’s look at the difference between the **flyto(X,Y);** and the **leapto(X,Y)** methods. In the image below, both Pip and Flup have started from (10,0) and have moved to (60,40). Pip who uses the **flyto(X,Y);** method moves along a straight line, from (10,0) to (60,40). Flup, who has used a **leapto(X,Y);** has executed a parabolic arc. But it ends at (60,40). It is a bit like **hopTo(X);** but while the latter will always return the Actor to the ground, you can use **leapto(X,Y);** to jump on top of scenery.



The Differences between Actors, Props and Scenery.

1) You can add any number of scenery items (including several of the same name). Scenery cannot move.

2) You can add one of each type of Prop. Props can execute all the above methods except express emotions. So you can't use **is(…)** or **feels(…)**.

3) Actors are like Props, you can add just one of each type. They can express emotions, so you can write **pip.feels(emotion);** or **flup.is(emotion);** where the following emotions are available, e.g. **pip.feels(puzzled);**

content, happy, puzzled, sad, excited, scared, worried, angry, surprised

4) The names of Props are derived from the names of scenery items my adding "**my"** to the front. For example **barrel** is scenery, but **mybarrel** is a prop. But not all scenery is a prop!

The Configuration File

This is the file **Header.txt** in the root directory. You can open this with a text editor (Notepad, Notepad++). There are two lines which you can change. **Take care** if you change this file; any error then the engine will be unforgiving. I suggest you make a backup, copy and rename to **HeaderBak.txt**.

1) The line

canvas.animationTime=2.0;

sets the time each method action takes, here to 2.0 seconds. To speed things up, you could reduce this to 1.5, or even something smaller. But too small a value may make the engine crash.

2) There are two "commented out" lines. You can un-comment one or the other to change the font size

canvas.gui.selectedSourcePanel.setFontSize(16);

or the font family, type and size

canvas.gui.selectedSourcePanel.setFont("Ariel",Font.BOLD,14);

Importing Children's Assets

The engine really becomes alive when children create their own Props and Backgrounds, each of which needs a single image:

1) A background image must have size width = 900 pels, height = 600 pels, and it must be a .jpg image. So if a child creates an image **myImage.jpg** then they can use it by a call to

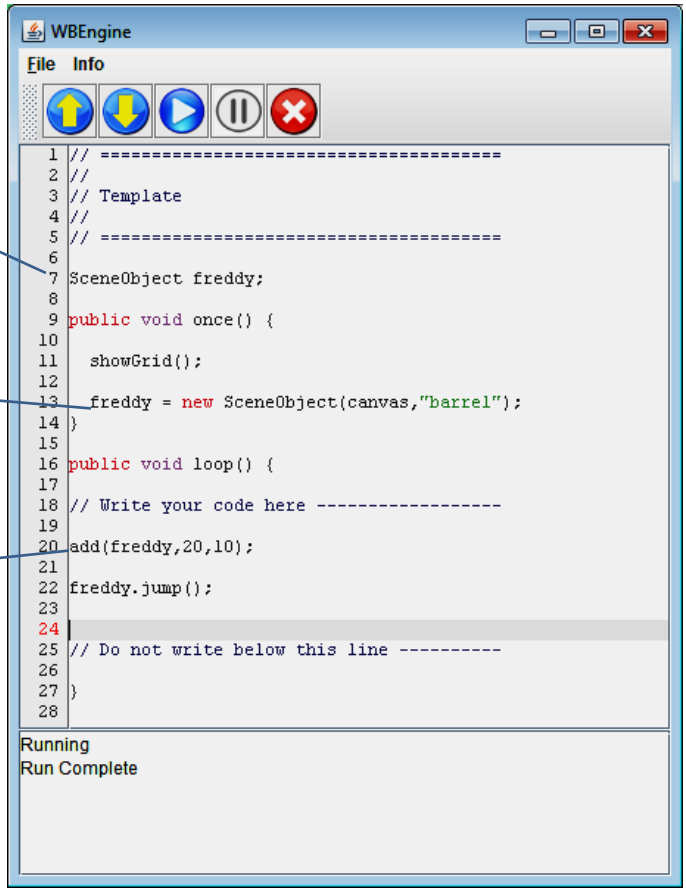
setScene("myImage"); or **clearScene("myImage");**

2) Prop images can vary in size, look at the images in the **data** folder to get an idea. Importing them is a little more tricky. Here's an example. The declaration must go at the top. The creation must go in the **once()** body and it can be used, as normal within the **loop()** body. Note that you use it by referring to its **name** (here "freddy") rather than the image name. This must be made clear to the children.

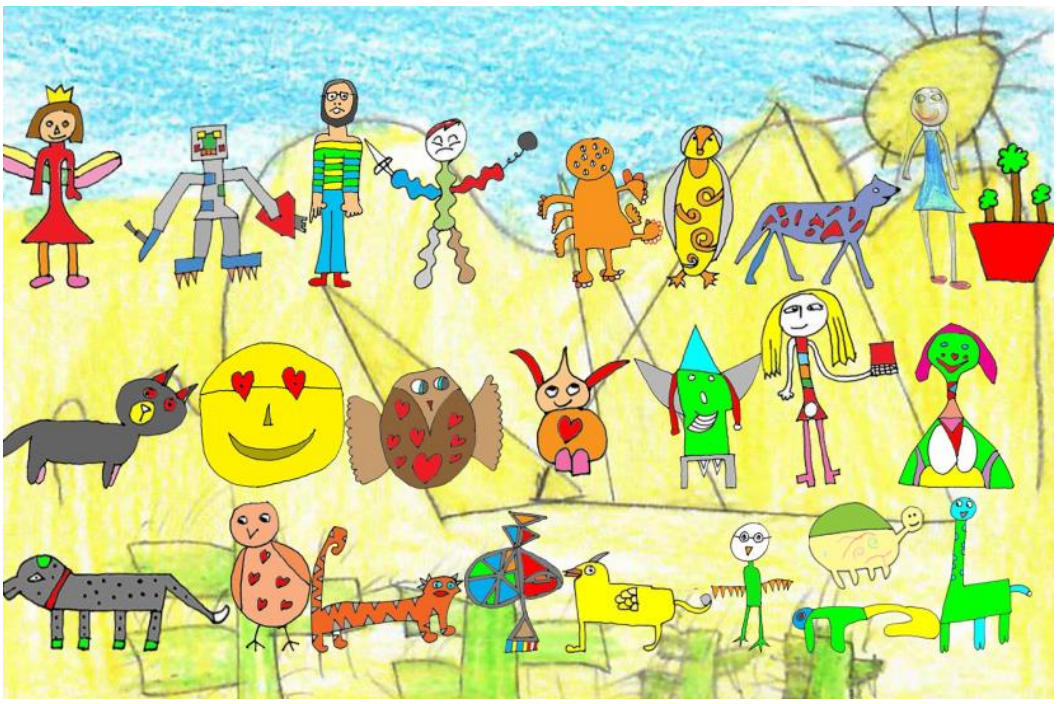
Declare a SceneObject and give it any **name** (here “freddy”)

Create the **named** object, using the image name – I chose “barrel”

Now you can use it, by its **name**



But there’s more than the technical side of introducing children’s own assets. I’m thinking about their design. The WeeBee characters, scenery and backgrounds have been designed *together* to produce *coherent* animations; it would not look good to have a humanoid character who is then made to fly around. Humans do not have the *affordance* of being able to fly (nor does Grog). So in designing their assets, children should think about how they relate to the code methods (in the table above). Can my asset fly? think? speak? Can it pick something up? Can it be picked up? My experience has been interesting, especially working with a mixed Yr3&4 group, below is a selection of their creations.

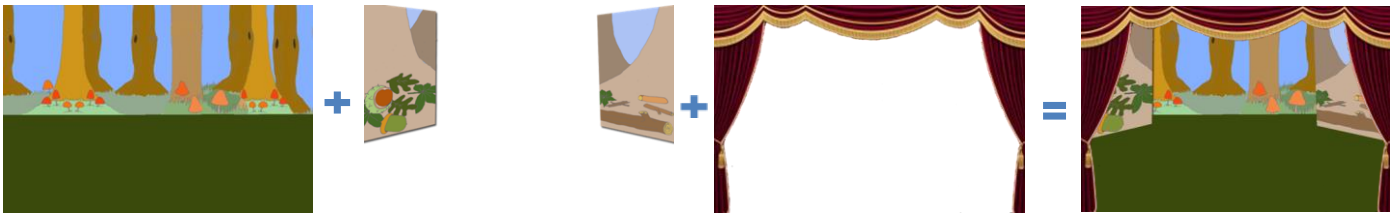


There was a clear gender difference. Boys created robots, robot warriors, and rather abstract zoomorphic characters. Girls produced more realistic animals which tended to be very benevolent. Then there were

anthropomorphic characters; a fairly and the class teacher (she's the one holding a laptop), I'm the one with a stripy jumper. Children's personal experiences were clearly at play here. I think there is a great opportunity to connect coding, English and art; how to design characters which can interact and tell a story. It all hinges on the *affordances* of everything in the scene.

Backgrounds : Theatre Backdrop and Flats

A small selection of backdrops and flat are available, to allow the coding of a theatrical performance, in effect children will be writing a play-script. The backdrops are 900 x 600 jpeg files, so can be called up using **setScene(…);** and the flats are matched .png files. This means the flats can be configured as props (as explained above) and made to move, i.e. descend from above, representing scene changes. Here's an example of a backdrop, the associated flat and how they appear combined, and finally with the façade added. Of course additional flats can be



created to provide scene-changes and entire theatrical performances. So far we have not trialled this approach with children.

Using Sounds

Sound effects, music or spoken narration can be placed in the folder **sounds** using the **.wav** file format. There are two ways sounds can be used:

1) A Long piece of music or narration can be started using e.g., **pip.sings("filename");** The sound will start when this line of code is executed, and will continue for the length of the sound file in seconds. This will therefore accompany the commands which follow.

2) A sound-effect can be played as part of the normal sequence of character actions, e.g., **pip.chirps("filename");** The sound-effect should last no more than 2-seconds. So in the following sequence

```
pip.jump();
pip.chirps("Egg");
pip.spin();
```

Pip will jump, then you will hear the sound of a cracking egg, then Pip will spin.

There is another way of using **chirps();** which will combine a sound with an action. This is shown in the following sequence

```
pip.jump();
pip.chirps("Egg",true);
pip.spin();
```

Here, Pip will jump, then she will spin accompanied by the sound of the cracking egg.