Make your own arcade games

ArcadeEngine and Runtime Revolution make creating computer games easier than ever

WHAT YOU'LL NEED

- *Revolution Studio* or *Media* (demo of *Studio* on DVD)
- *ArcadeEngine 2.0* (demo on DVD)
- The *MacFormat* DVD
- A graphics program that has the ability to export PNG files with transparent backgrounds, such as *Photoshop*, *The GIMP* or *GraphicConverter*

Mac FORMAT On the DVD

Download *Revolution Studio* and *ArcadeEngine* using this DVD

Computer games are all built around a few simple principles: objects appear on the screen, they move about, and they interact with one another. This holds true whether you're talking about archaic games like *Breakout* or the latest version of *Quake*.

In this tutorial, we'll be creating a program similar to the popular arcade game *Frogger*. In our game, the hero, Mr Frog, will need to cross a busy road to get to the safety of the pond. Put into terms of game mechanics, we will have one object, Mr Frog, that moves in a vertical direction, and a number of other objects, the cars on the road, that move in a horizontal direction. The player will control the movement of Mr Frog, while the computer will move the cars. If the computer detects a collision between Mr Frog and the cars, the player loses, but if the player gets Mr Frog safely to the pond without a collision, the player wins. Essentially, the game is about moving objects and detecting collisions.

The right tools

ArcadeEngine is an add-on for the *Revolution* programming environment that provides a suite of intuitive, easily learned tools that handle the types of movements and interactions common to arcade-style games. For example, a straight-line movement handler, moveLinear, is defined by a series of custom properties that can be applied to objects such as graphics. Two important custom properties are startpoint and endpoint, which define where the starting point and end point of an object's path in terms of X and Y coordinates (based on the centre of the object relative to the top left corner of the card). Another custom property is step, which is the speed at which an object moves along its path. By raising or lowering the value of step, an object can be sped up or slowed down. A fourth custom property, isDistance, describes how far an object has moved along that path, and by changing the value of isDistance, an object can be moved instantly from one point to another. What *ArcadeEngine* is effectively doing is reducing the complexity of creating an animation into simply changing custom property values.

> "The player will control the movement of Mr Frog, while the computer will move the cars"

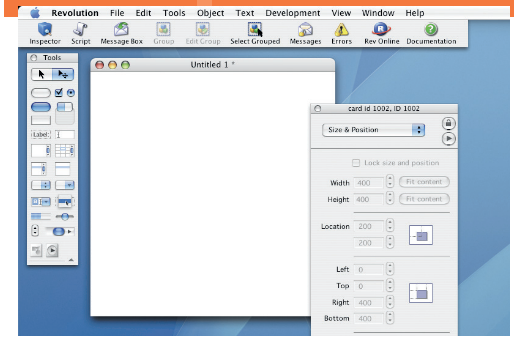
Collision effects

ArcadeEngine handles collisions by using another handler, imageCollide, which detects when the pixels making up one graphic overlap the pixels of another. This could be used, for example, to detect when a spaceship runs into an asteroid or a missile hits a monster. In this example, imageCollide will be used to detect collisions between Mr Frog and the cars.

Scripting is the most difficult part of programming *Revolution*. This tutorial explains what's going on in broad-brush terms, but to really get the hang of things, you'll need to read the *Revolution* and *ArcadeEngine* multimedia tutorials and documentation. In the meantime, you'll find the finished stack as well as the scripts and graphics used on the *MacFormat* DVD. Playing around with the code is a good way to get started with *Runtime Revolution*.

Neale Monks

STEP BY STEP Learning your way



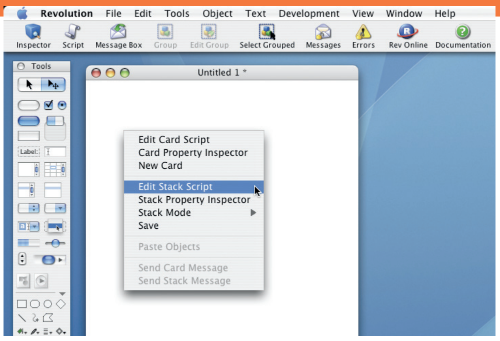
card id 1002, ID 1002

Size & Position

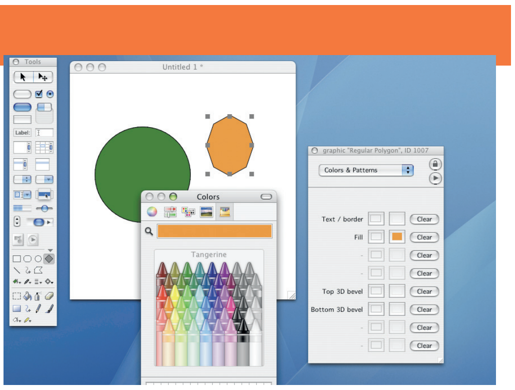
□ Lock size and position

Width 400 Fit content
Height 400 Fit content
Location 200
200
Left 0
Top 0
Right 400
Bottom 400

01 The **Properties** palette (by default to the left of the document) is where interface items are configured. A pull-down menu gives quick access to different groups of settings, such as colour, size and geometry.

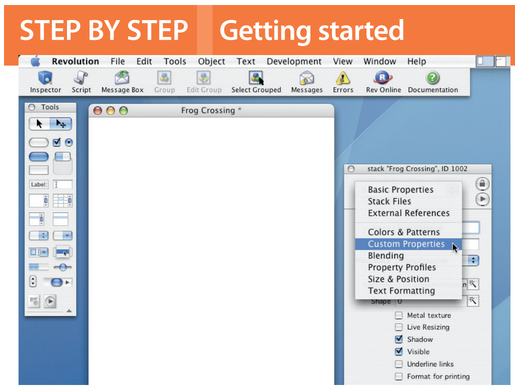


02 Control-clicking an item brings up a contextual menu, enabling quick access to an item's script. A button, for example, will have a script that tells it what to do when the user clicks that button.

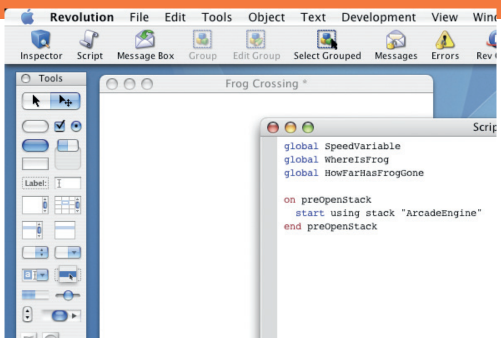


03 Interface objects are added using the **Tool** palette (by default, on the left of the document). Buttons, text fields and other controls are dragged from the palette on to the card. A disclosure triangle reveals the graphical tools, which work much like those on other Mac programs. Colours, sizes, and so on are set using the **Properties** palette.

STEP BY STEP Getting started

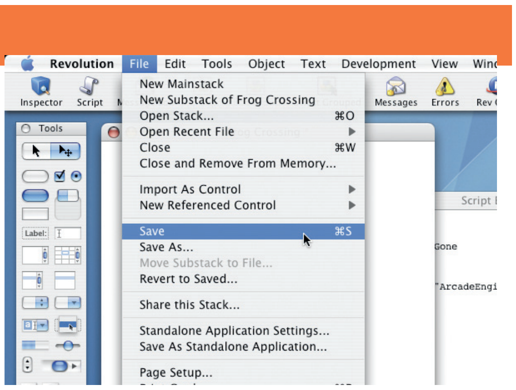


01 Create a new stack. Under the **Basic Properties** section of the **Properties** palette, give it a suitable name (in this case **"Frog Crossing"**) and then resize the stack in the **Size and Position** section (in this case 400 pixels by 600 pixels). Control-click the stack and in the contextual menu that appears, select **Edit Stack Script**.


Revolution File Edit Tools Object Text Development View Wind
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev
Tools
Label:
Frog Crossing *
Scrip

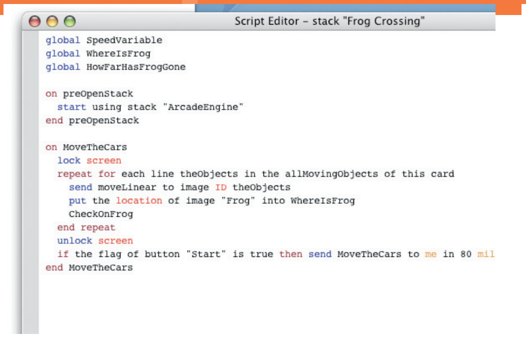
```
global SpeedVariable
global WhereIsFrog
global HowFarHasFrogGone

on preOpenStack
  start using stack "ArcadeEngine"
end preOpenStack
```

02 The stack script is where code goes that you want accessible to the entire stack. Open the file stack script.txt on the *MacFormat* DVD; this contains all the code you'll need for this project. For now, add Sections 1 and 2; these create some variables used later on and activate *ArcadeEngine* each time the stack is opened.


Revolution File Edit Tools Object Text Development View Wind
Inspector Script Messages Errors Rev
Tools
Label:
New Mainstack
New Substack of Frog Crossing
Open Stack... ⌘O
Open Recent File ▶
Close ⌘W
Close and Remove From Memory...
Import As Control ▶
New Referenced Control ▶
Save ⌘S
Save As...
Move Substack to File...
Revert to Saved...
Share this Stack...
Standalone Application Settings...
Save As Standalone Application...
Page Setup...
Script
Gone
"ArcadeEngi

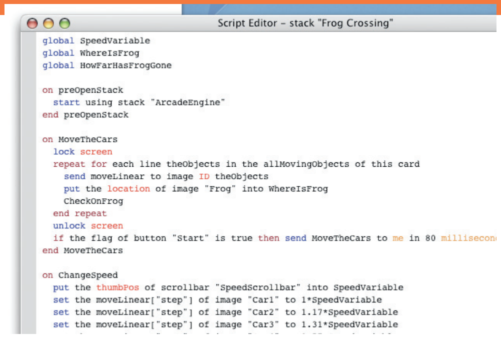
03 When you're done, press the **Apply** button at the foot of the scripting window, and then close the window. To get *ArcadeEngine* working, save, close and then re-open the stack.

STEP BY STEP Adding commands


Script Editor – stack "Frog Crossing"

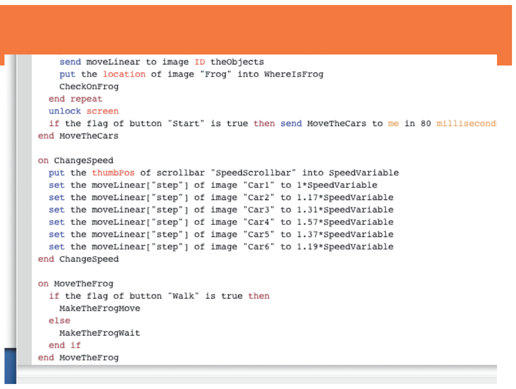
```
global SpeedVariable
global WhereIsFrog
global HowFarHasFrogGone

on preOpenStack
  start using stack "ArcadeEngine"
end preOpenStack

on MoveTheCars
  lock screen
  repeat for each line theObjects in the allMovingObjects of this card
    send moveLinear to image ID theObjects
    put the location of image "Frog" into WhereIsFrog
    CheckOnFrog
  end repeat
  unlock screen
  if the flag of button "Start" is true then send MoveTheCars to me in 80 mil
end MoveTheCars
```

01 Re-open the stack script, and add Section 3. This command moves a list of objects along straight-line paths. Locking and unlocking the screen between each 80-millisecond frame makes the animation run more smoothly. Note the use of a repeat-loop. This is much faster than moving each graphic with a separate line of code.


Script Editor – stack "Frog Crossing"

```
global SpeedVariable
global WhereIsFrog
global HowFarHasFrogGone

on preOpenStack
  start using stack "ArcadeEngine"
end preOpenStack

on MoveTheCars
  lock screen
  repeat for each line theObjects in the allMovingObjects of this card
    send moveLinear to image ID theObjects
    put the location of image "Frog" into WhereIsFrog
    CheckOnFrog
  end repeat
  unlock screen
  if the flag of button "Start" is true then send MoveTheCars to me in 80 millisecon
end MoveTheCars

on ChangeSpeed
  put the thumbPos of scrollbar "SpeedScrollbar" into SpeedVariable
  set the moveLinear["step"] of image "Car1" to 1*SpeedVariable
  set the moveLinear["step"] of image "Car2" to 1.17*SpeedVariable
  set the moveLinear["step"] of image "Car3" to 1.31*SpeedVariable
```

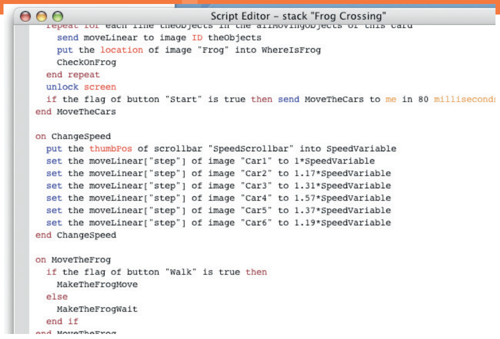
02 Now add Section 4 of the code. Later on, we'll be adding a scrollbar slider to control the speed of the game. When the slider is moved it will send its value ("thumbPos") to the **ChangeSpeed** command, which will in turn raise or lower the speed at which the cars move along the road.



```
    send moveLinear to image ID theObjects
    put the location of image "Frog" into WhereIsFrog
    CheckOnFrog
  end repeat
  unlock screen
  if the flag of button "Start" is true then send MoveTheCars to me in 80 millisecond
end MoveTheCars

on ChangeSpeed
  put the thumbPos of scrollbar "SpeedScrollbar" into SpeedVariable
  set the moveLinear["step"] of image "Car1" to 1*SpeedVariable
  set the moveLinear["step"] of image "Car2" to 1.17*SpeedVariable
  set the moveLinear["step"] of image "Car3" to 1.31*SpeedVariable
  set the moveLinear["step"] of image "Car4" to 1.57*SpeedVariable
  set the moveLinear["step"] of image "Car5" to 1.37*SpeedVariable
  set the moveLinear["step"] of image "Car6" to 1.19*SpeedVariable
end ChangeSpeed

on MoveTheFrog
  if the flag of button "Walk" is true then
    MakeTheFrogMove
  else
    MakeTheFrogWait
  end if
end MoveTheFrog
```

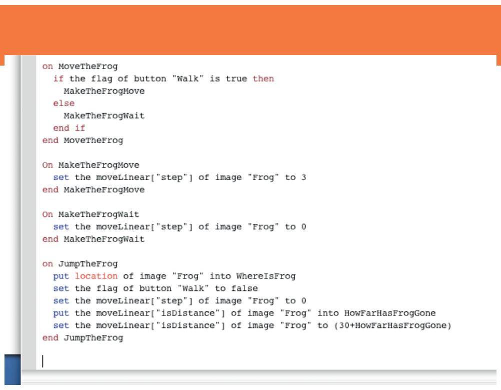
03 Besides the cars, the frog needs to move, too. By adding Section 5 to the stack script, we're preparing for a button that will either start or stop the frog's movement across the road. Note the use of the if-else option, a key method for dealing with situations where more than one thing might happen.

STEP BY STEP Controlling the frog


Script Editor – stack "Frog Crossing"

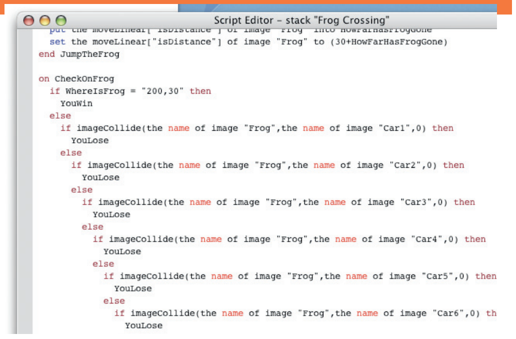
```
  repeat for each line theObjects in the allMovingObjects of this card
    send moveLinear to image ID theObjects
    put the location of image "Frog" into WhereIsFrog
    CheckOnFrog
  end repeat
  unlock screen
  if the flag of button "Start" is true then send MoveTheCars to me in 80 milliseconds
end MoveTheCars

on ChangeSpeed
  put the thumbPos of scrollbar "SpeedScrollbar" into SpeedVariable
  set the moveLinear["step"] of image "Car1" to 1*SpeedVariable
  set the moveLinear["step"] of image "Car2" to 1.17*SpeedVariable
  set the moveLinear["step"] of image "Car3" to 1.31*SpeedVariable
  set the moveLinear["step"] of image "Car4" to 1.57*SpeedVariable
  set the moveLinear["step"] of image "Car5" to 1.37*SpeedVariable
  set the moveLinear["step"] of image "Car6" to 1.19*SpeedVariable
end ChangeSpeed

on MoveTheFrog
  if the flag of button "Walk" is true then
    MakeTheFrogMove
  else
    MakeTheFrogWait
  end if
end MoveTheFrog
```

01 Controlling whether the frog moves or stops is its moveLinear step value. The pair of commands in Sections 6 and 7 changes this value. Depending on the "flag" of the walk button, one or other of those commands will be triggered. A flag is like a checkbox that's toggled on or off each time the button is pressed.



```
on MoveTheFrog
  if the flag of button "Walk" is true then
    MakeTheFrogMove
  else
    MakeTheFrogWait
  end if
end MoveTheFrog

On MakeTheFrogMove
  set the moveLinear["step"] of image "Frog" to 3
end MakeTheFrogMove

On MakeTheFrogWait
  set the moveLinear["step"] of image "Frog" to 0
end MakeTheFrogWait

on JumpTheFrog
  put location of image "Frog" into WhereIsFrog
  set the flag of button "Walk" to false
  set the moveLinear["step"] of image "Frog" to 0
  put the moveLinear["isDistance"] of image "Frog" into HowFarHasFrogGone
  set the moveLinear["isDistance"] of image "Frog" to (30+HowFarHasFrogGone)
end JumpTheFrog
```

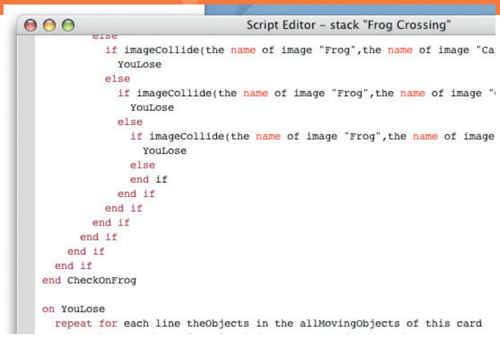
02 Besides walking, our frog can jump as well. The command in Section 8 puts how far the frog has gone (the isDistance of moveLinear) into a variable, adds 30 to it, and then resets the position of the Frog back to where it was.


Script Editor – stack "Frog Crossing"

```
  put the moveLinear["isDistance"] of image "Frog" into HowFarHasFrogGone
  set the moveLinear["isDistance"] of image "Frog" to (30+HowFarHasFrogGone)
end JumpTheFrog

on CheckOnFrog
  if WhereIsFrog = "200,30" then
    YouWin
  else
    if imageCollide(the name of image "Frog",the name of image "Car1",0) then
      YouLose
    else
      if imageCollide(the name of image "Frog",the name of image "Car2",0) then
        YouLose
      else
        if imageCollide(the name of image "Frog",the name of image "Car3",0) then
          YouLose
        else
          if imageCollide(the name of image "Frog",the name of image "Car4",0) then
            YouLose
          else
            if imageCollide(the name of image "Frog",the name of image "Car5",0) then
              YouLose
            else
              if imageCollide(the name of image "Frog",the name of image "Car6",0) th
                YouLose
```

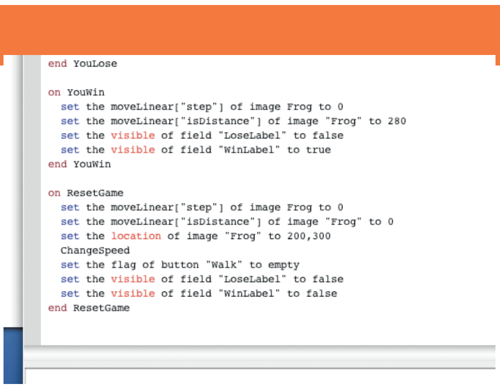
03 Section 9 of the code checks whether the frog has arrived at its goal, in which case the player wins, or if the frog has been hit by one of the cars, via a nested series of if-else options, in which case the player loses. Note the use of co-ordinates to define the "winning" position on the card.

STEP BY STEP Finishing off the script


Script Editor – stack "Frog Crossing"

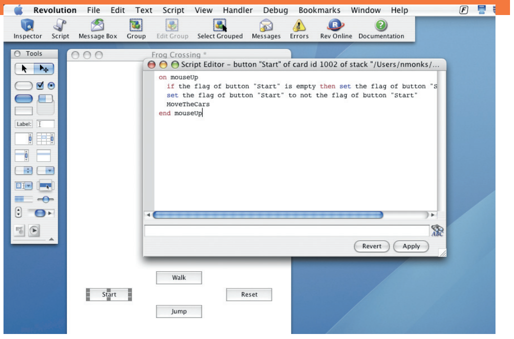
```
        else
          if imageCollide(the name of image "Frog",the name of image "Ca
            YouLose
          else
            if imageCollide(the name of image "Frog",the name of image "
              YouLose
            else
              if imageCollide(the name of image "Frog",the name of image
                YouLose
              else
              end if
            end if
          end if
        end if
      end if
    end if
  end if
end CheckOnFrog

on YouLose
  repeat for each line theObjects in the allMovingObjects of this card
```

01 A couple of additional commands (Sections 10 and 11) stop the game and switch on appropriate labels on the game declaring whether the player has won or lost. When an item's visibility is true, it is visible; if false, it is hidden.



```
end YouLose

on YouWin
  set the moveLinear["step"] of image Frog to 0
  set the moveLinear["isDistance"] of image "Frog" to 280
  set the visible of field "LoseLabel" to false
  set the visible of field "WinLabel" to true
end YouWin

on ResetGame
  set the moveLinear["step"] of image Frog to 0
  set the moveLinear["isDistance"] of image "Frog" to 0
  set the location of image "Frog" to 200,300
  ChangeSpeed
  set the flag of button "Walk" to empty
  set the visible of field "LoseLabel" to false
  set the visible of field "WinLabel" to false
end ResetGame
```

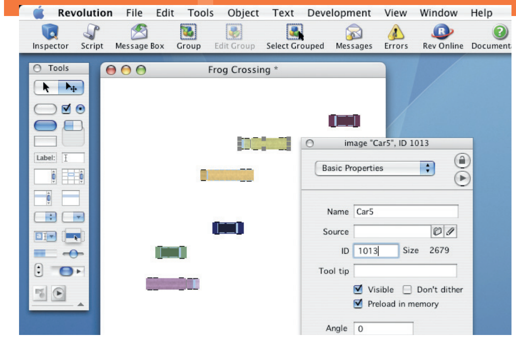
02 A fourth button will provide a way to reset the game, sending the frog back to the start, so the player can try again. Section 12 takes care of this by hiding the win/lose labels and resetting the location of the frog (again, using co-ordinates).


Revolution File Edit Text Script View Handler Debug Bookmarks Window Help
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev Online Documentation
Frog Crossing *
Revert
Apply
Walk
Start
Reset
Jump
Script Editor – button "Start" of card id 1002 of stack "/Users/nmonks/...

```
on mouseUp
  if the flag of button "Start" is empty then set the flag of button "S
  set the flag of button "Start" to not the flag of button "Start"
  MoveTheCars
end mouseUp
```

03 Four buttons need to be placed on the bottom half of the card as shown here, and renamed **Start**, **Walk**, **Jump** and **Reset**. Open the folder on the DVD entitled **Control Scripts**. Inside, you'll find scripts for each of these buttons. Control-click a button to edit its script, and then paste the required code. As far as coding goes, we're almost done!

STEP BY STEP Making the animation


Revolution File Edit Tools Object Text Development View Window Help
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev Online Document
Tools
Frog Crossing *
image "Car5", ID 1013

Basic Properties

Name: Car5

Source:

ID: 1013 Size 2679

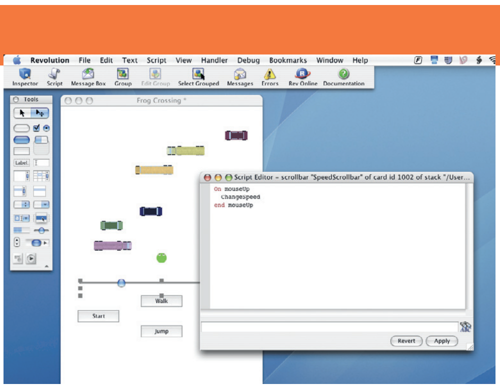
Tool tip:

☑ Visible ☐ Don't dither

☑ Preload in memory

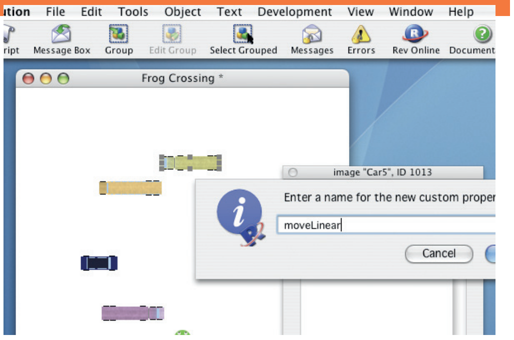
Angle: 0

01 In a graphics program, create six vehicles 75x20 pixels in size. Save them as PNG files, ensuring that they have transparent backgrounds. Go back to *Revolution*, and under **File**, choose **Import As Control**, and import the images. Using the basic properties section of the **Properties** palette, rename them **Car1**, **Car2**, etc, and note down their ID numbers.


Revolution File Edit Text Script View Handler Debug Bookmarks Window Help
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev Online Documentation
Frog Crossing *
Start
Walk
Jump
Revert
Apply
Script Editor – scrollbar "SpeedScrollbar" of card id 1002 of stack "/User...

```
On mouseUp
  ChangeSpeed
end mouseUp
```

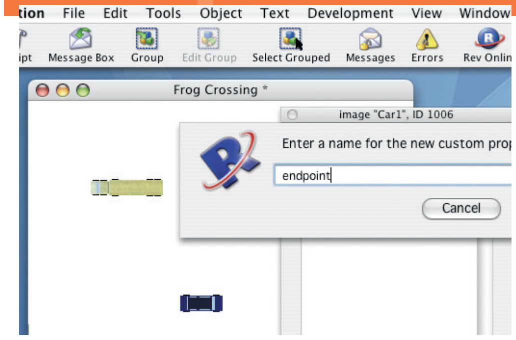
02 Make a 20x20 pixel Frog, name it **Frog**, and note its ID. Place it beneath the cars. Drag a scrollbar from the **Tool** palette on to the card, resize it, and edit its properties so that it is called **SpeedScrollbar**, has a maximum value of 20, and the "show value" box is unchecked. Edit its script, using the script scrollbar script.txt inside the **Control Scripts** folder on the DVD.


ution File Edit Tools Object Text Development View Window Help
ript Message Box Group Edit Group Select Grouped Messages Errors Rev Online Document
Frog Crossing *
image "Car5", ID 1013
Cancel
Enter a name for the new custom proper

moveLinear

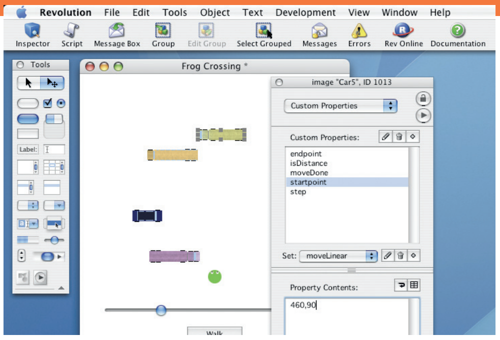
03 For each car, use the **Properties** palette to find its custom properties. Halfway down are the existing "sets". We don't want to edit these, so click on the plus sign in this row to add a new set, and call it **moveLinear**. Do this for the frog as well.

STEP BY STEP Custom properties


tion File Edit Tools Object Text Development View Window
ipt Message Box Group Edit Group Select Grouped Messages Errors Rev Onlin
Frog Crossing *
image "Car1", ID 1006
Cancel
Enter a name for the new custom prop

endpoint

01 Custom properties contain information unique to an item; in this case, movement. At the custom properties section, choose the moveLinear set, and then click on the plus sign to add each custom property. These are called "startpoint" and "endpoint". Enter their values into the property contents field: **-60,270** and **460,270** for these respectively.


Revolution File Edit Tools Object Text Development View Window Help
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev Online Documentation
Tools
Frog Crossing *
Walk
image "Car5", ID 1013

Custom Properties

Custom Properties:

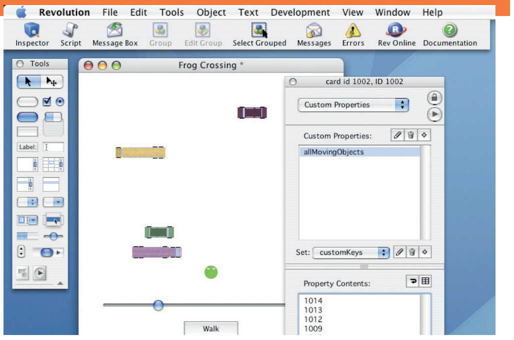
endpoint
isDistance
moveDone
startpoint
step

Set: moveLinear

Property Contents:

460,90

02 Repeat this for the other cars. The required values are: Car 2, **-60,240** and **460,240**; Car3, **-60,210** and **460,210**; Car4, **460,120** and **-60,120**; Car5, **460,90** and **-60,90**; and Car6, **460,60** and **-60,60**. These numbers define the start and ending points of the movement of each car.


Revolution File Edit Tools Object Text Development View Window Help
Inspector Script Message Box Group Edit Group Select Grouped Messages Errors Rev Online Documentation
Tools
Frog Crossing *
Walk
card id 1002, ID 1002

Custom Properties

Custom Properties:

allMovingObjects

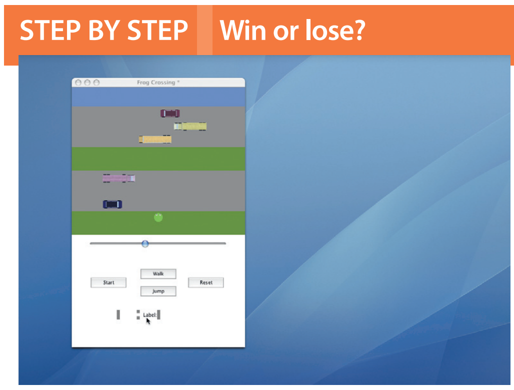
Set: customKeys

Property Contents:

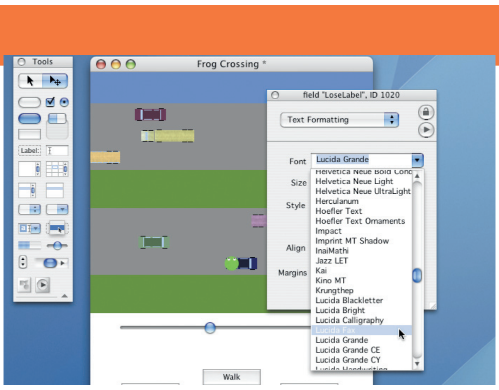
1014
1013
1012
1009

03 Give the frog a moveLinear set of custom properties, and set its startpoint and endpoint to **200,300** and **200,30**. Control-click the stack, select **Card Property Inspector**, and select the **Custom Properties**. Add a custom property to the customKeys property set, and call it **allMovingObjects**. In **property contents** field, enter the IDs of the cars and frog.

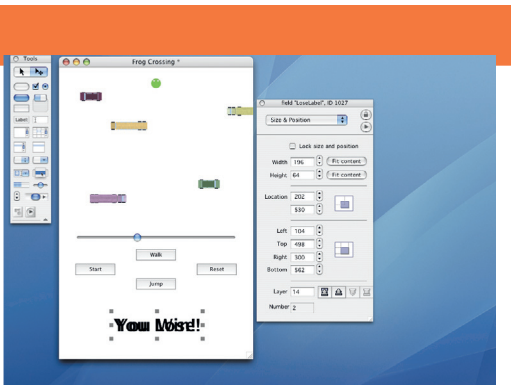
STEP BY STEP Win or lose?



01 We need a way to tell the player whether they've won or lost. *Revolution* has a variety of multimedia tools, so we could play a sound effect or run an animation of some kind, but to keep things easy, we're simply going to reveal a bit of text. Begin by dragging two label fields from the **Tool** palette on to the card.

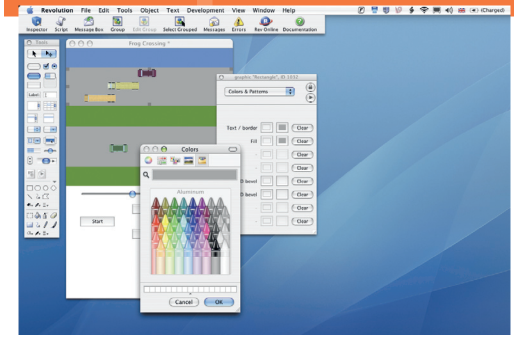


02 Use the **Properties** palette to name one **WinLabel** and the other **LoseLabel**. At the contents section of the **Properties** palette, type in an appropriate message for each one, such as **"You Lose!"**, and then use the text formatting section to style the text correctly.

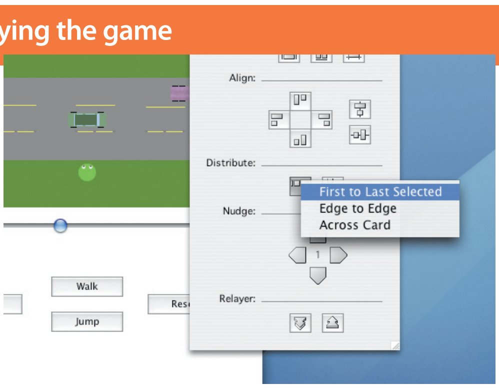


03 Position the two messages on the card, one on top of each other, as shown here. Don't worry that it looks a bit messy: the stack script hides both of them to begin with, and shows only the right one depending on whether the player wins or loses.

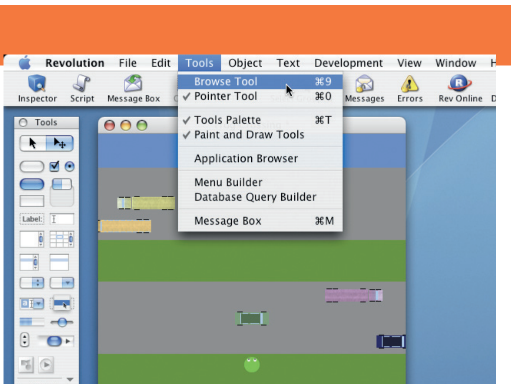
STEP BY STEP Finishing touches and playing the game



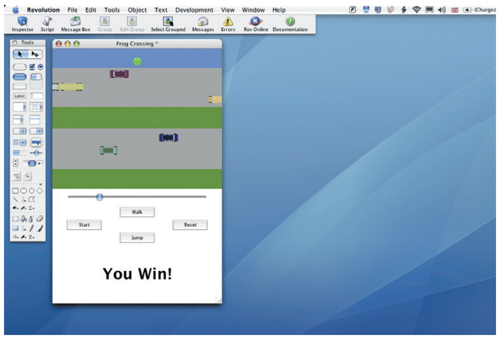
01 *Revolution* has some basic graphic tools that can be used to enhance the animation. Use coloured rectangles to create grassy verges, the roads and the pond. Use the **Layer** buttons on the **Size and Position** section of the **Properties** palette to position them underneath the Frog and cars.



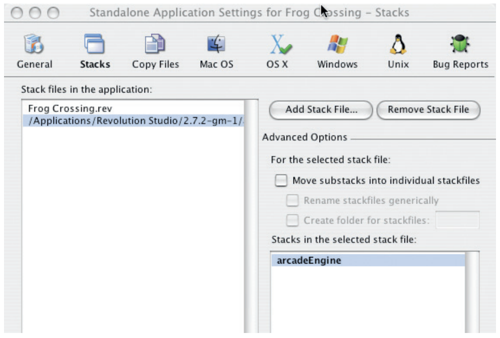
02 Add some yellow lines down the middle of the road to mark the lanes. When you select multiple objects, the **Properties** palette will include an **Align Objects** section, useful for aligning and spacing out graphics and other items.



03 You're done! You can run the stack by switching into **Browse** mode; do this by selecting the **Browse** tool under the **Tools** menu. If you want to go back and edit something in the stack, choose the **Pointer** tool.



04 To play the game, press the **Start** button and the animation will start running. Move the **slider** to change the speed of the cars. Use the **Walk** button to advance the frog, and the **Jump** button to hop out of trouble. When you're done, press the **Reset** button to start over.



Stack files in the application:
Frog Crossing.rev
/Applications/Revolution Studio/2.7.2-gm-1/

Advanced Options

For the selected stack file:

□ Move substacks into individual stackfiles

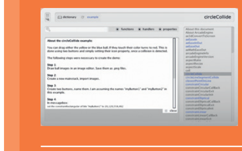
□ Rename stackfiles generically

□ Create folder for stackfiles:

Stacks in the selected stack file:
arcadeEngine

05 Stacks can be shared with others either as they are or by exporting them as standalone applications (you'll need *Revolution Studio* for that). If you choose to export as an application, be sure *ArcadeEngine* is included in the build via the **Standalone Application Settings** window. Without *ArcadeEngine*, the finished application won't work.

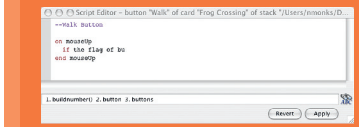
TOOL SCHOOL



ArcadeEngine comes with an interactive guide that contains lots of fully explained examples and short step-by-step tutorials.

EXPERT TIP

As you type in code, you'll see *Revolution* offer suggestions at the bottom of the window. To use these, press ⌘+[1] to get the first, ⌘+[2] to get the second, and so on. Using these will save you masses of time!


Script Editor – button "Walk" of card "Frog Crossing" of stack "/Users/nmonks/D...
Revert
Apply

```
--Walk Button

on mouseUp
  if the flag of bu
end mouseUp
```

1. buildnumber() 2. button 3. buttons