Alistair Stewart

The Official AppGameKit STUDIO Tutorial Guide Vol #2

Digital Skills

the GameCreators

The Official AppGameKit Studio Tutorial Guide
Volume 2

Alistair Stewart

Digital Skills
Milton
Barr
Ayrshire
KA26 9TY
UK

www.Digital-Skills.co.uk

www.digital-skills.co.uk
tel: +44(0)1465 861 638

First Published May 2019

Title: The Official AppGameKit Studio Tutorial Guide Vol#2

ISBN: 978-1-874107-25-5

Contents

3D Physics: Joints, Rag Doll, and Character Controllers

Using a Lan

Accessing a Server

The Previous Volume

What went Before

Book (Volume 1)

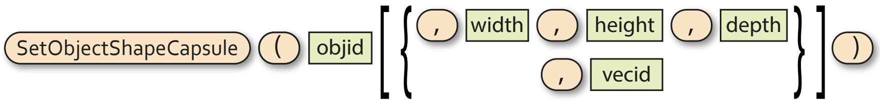
This is the second volume of the **Hands On AGK2 BASIC** text.

In the previous book we started with the very fundamentals of programming and travelled through the basics of control structures, arrays and functions. We also looked at various aspects of games programming covering images, sprites, animation, multimedia and using input devices.

Along the way we developed a core structure for all our programs and built a library of useful user-defied functions. And, of course, we programmed a few games!

Syntax Diagrams

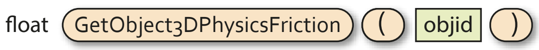
When describing the structure of an AGK2 statement or function call we made use of syntax diagrams such as that shown below.



- Terms within the orange-coloured capsules must be included exactly as shown.
- Terms in the green boxes signify variable names or fixed values the exact value of which are dependent on the programmer.
- Items within square brackets are optional and may be omitted.
- Items within braces offer a choice of structure (one option per row).

A description of what each programmer-defined value represents will be included beneath the syntax diagram.

Some syntax diagrams begin with the term integer, float or string to signify the type of value returned by a function:



Program Structure

For simple one-play games we developed a set of functions to be called which resulted in the following program structure:

```
// Project: project name
// Created: date created

//*** Include required library files ***
#include "../Function Library/Name of library file"
```

```
//*** User-defined types ***
type GameType
   main data elements of the program
endtype

//*** Global variables ***
global g as GameType

//*****************************************
//***           Main program          ***
//*****************************************
InitialiseScreen(width,height,"Title", background colour,
orientations allowed)
ShowSplashScreen("name of image file")
LoadResources()
HideSplashScreen(seconds before hide)
InitialiseGameVariables()
CreateInitialLayout()
do
   in = GetUserInput()
   HandleUSerInput(in)
   HandleOther()
   Sync()
loop
//*****************************************
//***          Main Functions         ***
//*****************************************
Functions defined here
```

The code required by the functions *InitialiseScreen()*, *ShowSplashScreen()* and *HideSplashScreen()* is defined within a library file (see below) since their logic is the same for every program.

The other functions named in the main program need to be coded within the *Main Functions* area of the code since the logic these execute will differ from app to app.

Libraries

Several library files were developed in the earlier volume. The libraries can be downloaded from this book's resources ZIP file. You'll find the relevant file at *AGK2_2/Resources/Ch00/Function Library*. This should be installed as a sub-folder off your main AGK2 working folder.

Details of the functions contained in the library (as well as the GUI library functions created in Chapter 4 of this book) are given in the PDF file *UDLibs.pdf* which can be found in the resources ZIP file under *AGK2_2/Resources/Ch00/UDLibs.pdf*.

Text Style

In this text code is shown in blue monospaced type:

```
in = GetUserInput()
HandleUSerInput(in)
HandleOther()
Sync()
```

as is any reference to AGK2-defined terms with the descriptive paragraphs.

Function names are always shown with terminating parentheses to distinguish them from variable names. AGK2 function names are in `blue monospaced` type while user-defined function names are shown in *italics*.

Important terms, when used for the first time are shown in **bold**.

Lines of code which, because of space restrictions, have had to be spread over two or more lines, start subsequent lines with the ↳ symbol – this symbol is not part of the code.

This text is designed primarily as a tutorial. As such many Activities build on work that has been carried out on previous pages, so jumping straight in to a specific section may sometimes make for a confusing read!

Downloads

Many of the Activities in this book require resources which you can download from the Digital Skills website (*www.digital-skills.co.uk*).

Comments

If you find an error or have any constructive comments on how we might improve this book please take the time to email us at

comments@digital-skills.co.uk

1

Tweening

In this Chapter:

❑ Tweening Concepts

❑ Sprite Tweening

❑ Text and Character Tweening

❑ Custom Tweening

❑ Chaining Tweens

Sprite Tweening

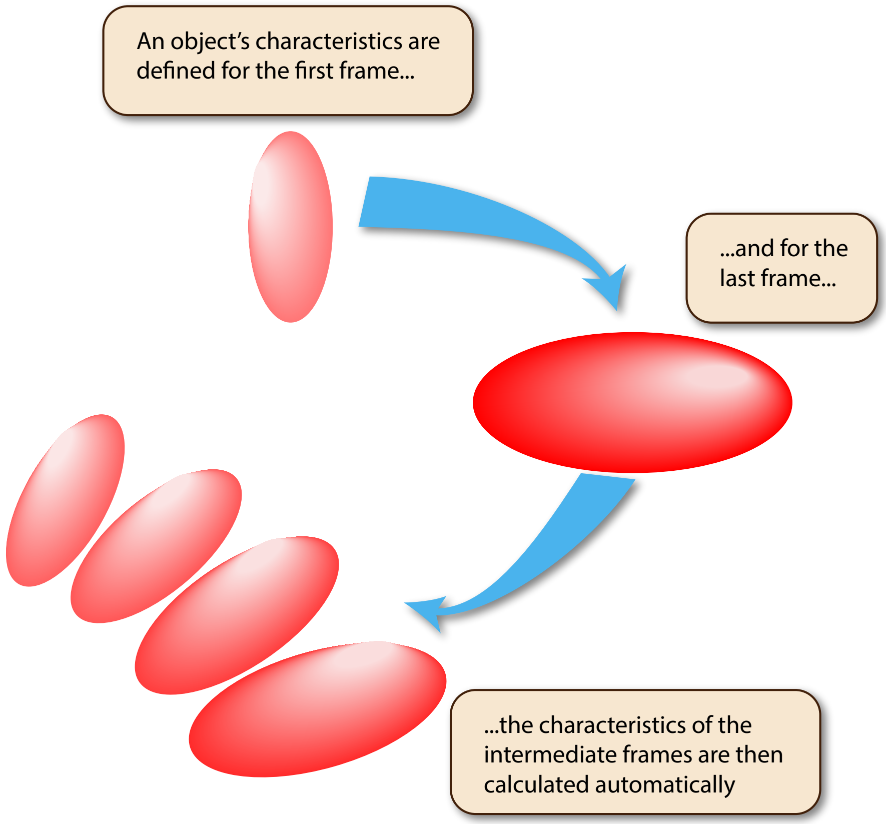
Introduction

The term **tweening** is a shortened version of **inbetweening** and is used, in a computer context, to describe the automatic manipulation of an object. Often this object will be a visible component such as a sprite or text resource.

With a bit of coding we can easily animate a sprite to move it from one location to another, rotate it, change its colour, resize it, or fade it from view. However, we can reduce the code required for these types of animations by having the program set up a **tween** in which the changes we want to apply to the sprite are specified and then leave it to that tween component to animate the whole process by automatically calculating and carrying out the intermediate steps between the start and finish states of the sprite (see FIG-1.1).

FIG-1.1

Tweening


An object's characteristics are defined for the first frame...
...and for the last frame...
...the characteristics of the intermediate frames are then calculated automatically

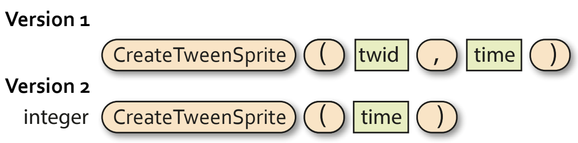
Sprite Tween Commands

CreateTweenSprite()

If we want to create a tween effect involving a sprite, we need to start by creating a tween object which will be used to hold details of the changes we want to apply to the sprite. The tween is created using the `CreateTweenSprite()` command (see FIG-1.2).

FIG-1.2

CreateTweenSprite()


Version 1
CreateTweenSprite (twid , time)
Version 2
integer CreateTweenSprite (time)

where:

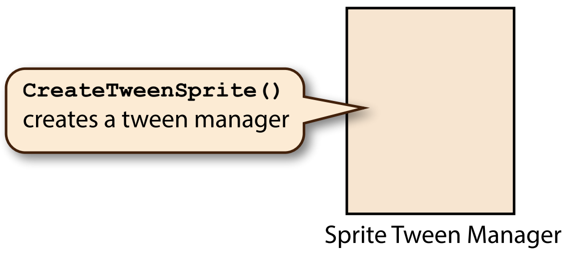
twid	is an integer value giving the ID to be assigned to the tween.
time	is a real value giving the nominal duration (in seconds) of the tween effect. The actual time taken to play is affected by other factors discussed later.

Version 1 of the command allows the user to select the ID assigned to the tween; version 2 assigns an ID automatically and returns the value assigned. Automatically assigned IDs start at 100001.

It might be useful to imagine the element we have just created as a tween "manager" into which we will later fit the effects we want to apply (see FIG-1.3).

FIG-1.3

A Tween Manager



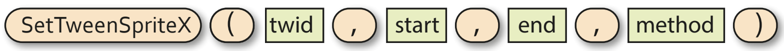
With the manager created, we now need to add the **behaviours** we require of the tween. These specify operations such as movement, rotation, resizing, opacity change and colour change.

SetTweenSpriteX()

If we want a tween to move a sprite horizontally, we can specify the start and end *x* coordinates of the tween as well as the movement style (detailed later) using the `SetTweenSpriteX()` command (see FIG-1.4).

FIG-1.4

SetTweenSpriteX()



where:

twid	is an integer value giving the ID of the tween manager.
start	is a real value giving the starting *x* coordinate of the sprite.
end	is a real value giving the final *x* coordinate of the sprite.
method	is an integer value giving the "style" of movement to be employed (these are described below). A value of -1 stops all tweening.

✎ **start** and **end** positions refer to the top left corner of the sprite.

method Functions

Rather than have to remember the integer value associated with each "style" of movement, AGK offers a set of helpfully named functions which return the appropriate integer value. These are shown in FIG-1.5.

FIG-1.5

Options for the *method* Parameter

Function Name	Numeric Value	Effect
TweenLinear()	0	Equally sized steps from start to finish
TweenSmooth1()	1	A gradual build up of speed at the start and slow down at the end
TweenSmooth2()	2	A faster build up and slow down than above
TweenEaseIn1()	3	A gradual build up of speed at the start
TweenEaseIn2()	4	A faster build up of speed at the start than above
TweenEaseOut1()	5	A gradual slow down at the end
TweenEaseOut2()	6	A faster slow down at the end than above
TweenBounce()	7	Creates a bounce effect at the end of the movement
TweenOvershoot()	8	Creates an overshoot effect at the end of the movement

We could set up our tween to gradually increase speed at the start of the movement and slow down at the end using the line:

```
SetTweenSpriteX(twid,0,100,TweenSmooth1())
```

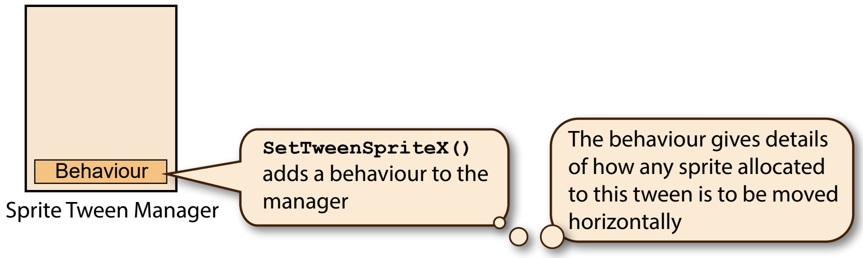
or, rather than use the method function name, substitute the numeric value:

```
SetTweenSpriteX(twid,0,100,1)
```

By making a call to `SetTweenSpriteX()` after having created our tween manager, we are, in effect, adding a behaviour to that manager (see FIG-1.6).

FIG-1.6

Adding a Behaviour to a Tween Manager

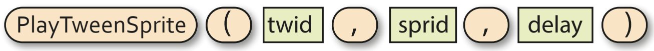

Behaviour
Sprite Tween Manager
SetTweenSpriteX() adds a behaviour to the manager
The behaviour gives details of how any sprite allocated to this tween is to be moved horizontally

PlayTweenSprite()

Before looking at other behaviours that can be added to a tween, let's link a sprite to our tween and see the effect created. The link between tween and sprite is achieved using the `PlayTweenSprite()` function (see FIG-1.7).

FIG-1.7

PlayTweenSprite()



where:

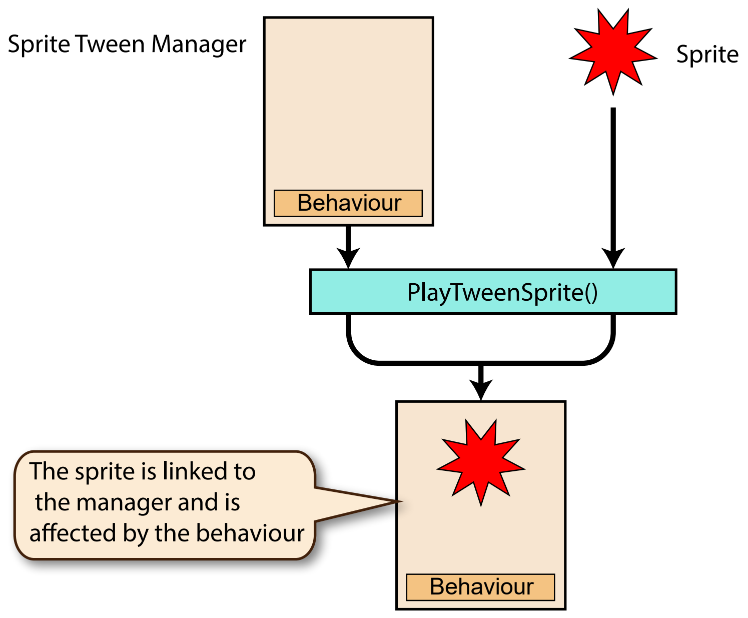
twid is an integer value giving the ID of the tween manager.

sprid	is an integer value giving the ID of the sprite to which the tween is to be applied.
delay	is a real number giving the delay (in seconds) before the tween should be applied.

The effect of calling `PlayTweenSprite()` is visualised in FIG-1.8.

FIG-1.8

Playing a Sprite Tween

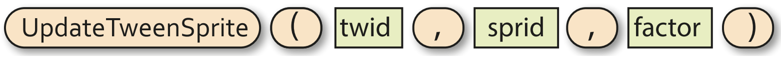

Sprite Tween Manager
Sprite
Behaviour
PlayTweenSprite()
The sprite is linked to the manager and is affected by the behaviour
Behaviour

UpdateTweenSprite()

There is one last step required in order to see the effects of our sprite tweening on the screen and that is to call the function `UpdateTweenSprite()` every time a new screen frame is created. This function has the format shown in FIG-1.9.

FIG-1.9

UpdateTweenSprite()



where:

twid	is an integer value giving the ID of the tween manager.
sprid	is an integer value giving the ID of the sprite to which the tween is to be applied.
factor	is a real number affecting the speed at which the animation created by the tweening is played. Lower values mean a slower speed.

When calling `UpdateTweenSprite()`, the usual value for *factor* is that returned by `GetFrameTime()`, so a typical call would be

```
UpdateTweenSprite(1,1,GetFrameTime()) //Tween 1, Sprite 1
```

The program below demonstrates the use of tweening by moving an oval-shaped sprite from the left to the right of the screen.

```
// Project: TweenSprite
// Created: 2016-11-04

//*** Include library files ***
#include "../Function Library/CoreLibrary.agc"

type GameType
   imgOval  as integer  //Oval image
   ovalID   as integer  //Oval sprite
   tweenID  as integer  //Sprite tween
endtype

global g as GameType

//****************************************
//***          Main program           ***
//****************************************
InitialiseScreen(1024,768,"Sprite Tweening", 0x686868, %1111)
LoadResources()
CreateInitialLayout()
do
   UpdateTweenSprite(g.tweenID,g.ovalID,GetFrameTime())
   Sync()
loop

//****************************************
//***           Functions              ***
//****************************************
function LoadResources()
   g.imgOval = LoadImage("oval.png")
endfunction


function CreateInitialLayout()
   //*** Create oval sprite ***
   g.ovalID = CreateSprite(g.imgOval)
   SetSpriteSize(g.ovalID,8,-1)
   SetSpritePosition(g.ovalID,10,50)
   //*** Create sprite tween ***
   g.tweenID = CreateTweenSprite(3.0)
   //*** Move sprite horizontally ***
   SetTweenSpriteX(g.tweenID,10,90,TweenSmooth1())
   //*** Play the tween ***
   PlayTweenSprite(g.tweenID,g.ovalID,1)
endfunction
```

Activity 1.1

Start a new project called *TweenSprite* and implement the code given above. Remember to copy the file *oval.png* from *AGK2/Resources/Ch01* to the project's *media* folder.
Run the program and observe the effect (notice how the sprite movement starts slowly and ends slowly.
The value returned by `GetFrameTime()` is approximately 0.016. In the call to `UpdateTweenSprite()`, change the last parameter to

a) 0.16 b) 0.008

Activity 1.1 (continued)

and see how this changes the speed of the movement.

Restore the parameter for `UpdateTweenSprite()` to its original value.

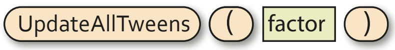
Run the program several more times, modifying the final parameter for `SetTweenSpriteX()` each time so that the effect of each option is observed.

UpdateAllTweens()

Where a program contains several tweens, rather than add a separate update call for each of these, we can update all tweens using `UpdateAllTweens()`. This function has the format shown in FIG-1.10.

FIG-1.10

UpdateAllTweens()



where:

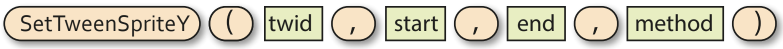
factor	is a real number affecting the speed at which the animation created by the tweening is played. Lower values mean a slower speed.

SetTweenSpriteY()

To create a tween which moves a sprite in the vertical direction we can use `SetTweenSpriteY()` (see FIG-1.11).

FIG-1.11

SetTweenSpriteY()



where:

twid	is an integer value giving the ID of the tween manager
start	is a real number giving the starting *y* coordinate of the sprite.
end	is a real number giving the final *y* coordinate of the sprite.
method	is an integer giving the "style" of movement to be employed. The options are the same as those for `SetTweenSpriteX()`.

✎ **start** and **end** positions refer to the top left corner of the sprite.

Activity 1.2

Modify *TweenSprite* by commenting out the call to `SetTweenSpriteX()` and add a call to `SetTweenSpriteY()` using the same parameters as applied to the horizontal movement.

Observe the effect of this change.

By commenting out the `SetTweenSpriteX()` call and replacing it with one to `SetTweenSpriteY()`, we have replaced one behaviour in our sprite tween manager with another. But there is no limit to the number of behaviours we may place in the manager, so it is quite possible to have both X and Y tween behaviours at the same time.

Activity 1.3

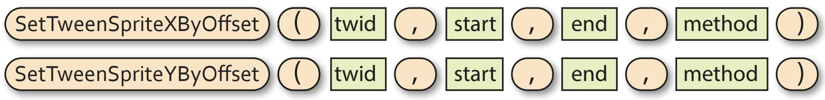
Modify *TweenSprite* removing the comment characters from the call to `SetTweenSpriteX()`.

Compile and run the program, observing how the two behaviours affect the movement of the sprite.

SetTweenSpriteXByOffset() and SetTweenSpriteYByOffset()

If we want to move a sprite relative to its centre (rather than its top-left corner), then we can use the `SetTweenSpriteXByOffset()` and `SetTweenSpriteYByOffset()` commands (see FIG-1.12).

FIG-1.12


SetTweenSpriteXByOffset (twid , start , end , method)
SetTweenSpriteYByOffset (twid , start , end , method)

SetTweenSpriteXByOffset()
SetTweenSpriteYByOffset()

where:

twid	is an integer value giving the ID of the tween manager
start	is a real number giving the starting *x* or *y* coordinate of the sprite.
end	is a real number giving the final *x* or *y* coordinate of the sprite.
method	is an integer giving the "style" of movement to be employed. The options are the same as for `SetTweenSpriteX()`.

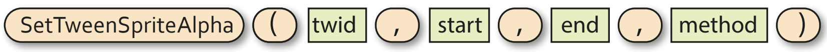
✎ **start** and **end** positions refer to the sprite offset.

By default, a sprite's offset is at its middle.

SetTweenSpriteAlpha()

To add a behaviour that adjusts the transparency of the sprite, we can use `SetTweenSpriteAlpha()` (see FIG-1.13).

FIG-1.13



SetTweenSpriteAlpha()

where:

twid	is an integer value giving the ID of the tween manager.
start	is an integer number (0 to 255) giving the starting opacity. (0:invisible; 255: opaque)
end	is an integer number (0 to 255) giving the final opacity.
method	is an integer (-1 to 8) giving the "style" of change in opacity to be employed.

The *method* options are comparable to those for movement. For example, option `TweenSmooth1()` will start with a slow change to the transparency, speed up, then slow down again as it nears the end value. Using `TweenBounce()` will cause the transparency to reach its end value, then regress slightly, until finally settling once again at the end value.

Activity 1.4

Modify *TweenSprite* by commenting out the tween's movement behaviours and adding one which cause the sprite to go from opaque to invisible.

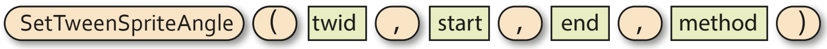
Observe how the tweening is affected by the different values for *method*

SetTweenSpriteAngle()

We can have the tween rotate its sprite using `SetTweenSpriteAngle()` (see FIG-1.14).

FIG-1.14

SetTweenSpriteAngle()



where:

twid	is an integer value giving the ID of the tween manager.
start	is a real number giving the starting angle in degrees.
end	is a real number giving the final angle in degrees. This value can be greater than 360 if we want the sprite to perform several rotations.
method	is an integer (-1 to 8) giving the "style" of the change in speed of rotation.

Activity 1.5

Modify *TweenSprite* by commenting out the tween's opacity behaviour and adding one which cause the sprite to rotate from 0^o to 720^o.

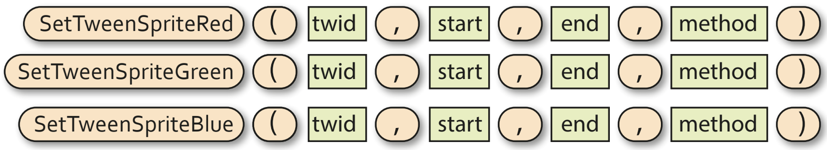
Observe how the tweening is affected by the different values for *method*.

SetTweenSpriteRed(), SetTweenSpriteGreen() and SetTweenSpriteBlue()

The colour components of a sprite can also be tweened using `SetTweenSpriteRed()`, `SetTweenSpriteGreen()`, and `SetTweenSpriteBlue()` (see FIG-1.15).

FIG-1.15

SetTweenSpriteRed()
SetTweenSpriteGreen()
SetTweenSpriteBlue()



where:

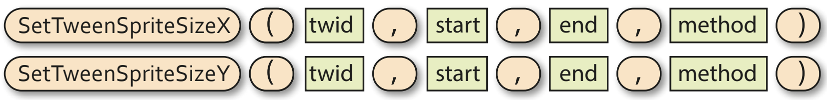
twid	is an integer value giving the ID of the tween manager.
start	is an integer value (0 to 255) giving the strength of the specified colour at the start of the tween.
end	is an integer value (0 to 255) giving the strength of the specified colour at the end of the tween.
method	is an integer (-1 to 8) detailing the technique to be used in the colour change.

SetTweenSpriteSizeX() and SetTweenSpriteSizeY()

To create tween behaviours that change a sprite's size, we can use `SetTweenSpriteSizeX()` (which modifies the height of the sprite) and `SetTweenSpriteSizeY()` (which changes its width). The syntax for each command is shown in FIG-1.16.

FIG-1.16

SetTweenSpriteX()
SetTweenSpriteY()


SetTweenSpriteSizeX (twid , start , end , method)
SetTweenSpriteSizeY (twid , start , end , method)

where:

twid	is an integer value giving the ID of the tween manager.
start	is an integer value (0 to 255) giving the size of the sprite at the start of the tween.
end	is an integer value (0 to 255) giving the size of the sprite at the end of the tween.
method	is an integer (-1 to 8) detailing the technique to be used when resizing the sprite.

Activity 1.6

Modify *TweenSprite* by removing the commenting forward slashes from existing statements.

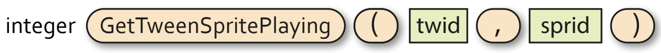
Make the sprite double in size as it moves from its start to end position and change its red component from 255 to zero.

GetTweenSpritePlaying()

To check if a sprite tween is currently running with a specified sprite, we can use `GetTweenSpritePlaying()` (see FIG-1.17).

FIG-1.17

GetTweenSpritePlaying()



where:

twid	is an integer value giving the ID of the tween manager.

sprid	is an integer value giving the ID of the sprite used within the tween manager.

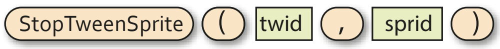
The function returns 1 if the tween is currently playing, otherwise zero is returned.

StopTweenSprite()

To stop a sprite tween playing, we can use `StopTweenSprite()`. This command has the format shown in FIG-1.18.

FIG-1.18

StopTweenSprite()



where:

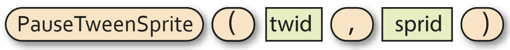
twid	is an integer value giving the ID of the tween manager.
sprid	is an integer value giving the ID of the sprite used within the tween manager.

PauseTweenSprite()

To pause a currently executing sprite tween, we can use `PauseTweenSprite()` (see FIG-1.19).

FIG-1.19

PauseTweenSprite()



where:

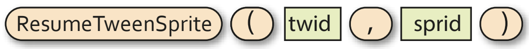
twid	is an integer value giving the ID of the tween manager.
sprid	is an integer value giving the ID of the sprite used within the tween manager.

ResumeTweenSprite()

To resume a currently paused sprite tween, we can use `ResumeTweenSprite()` (see FIG-1.20).

FIG-1.20

ResumeTweenSprite()



where:

twid	is an integer value giving the ID of the tween manager.
sprid	is an integer value giving the ID of the sprite used within the tween manager.

In the listing below, the *TweenSprite* project has been modified so that the tween pauses when the *Enter* key is first pressed and then resumes on a second press of the key.

```
// Project: TweenSprite
// Created: 2016-11-04
```

```
//*** Include library files ***
#include "../Function Library/CoreLibrary.agc"

type GameType
   imgOval  as integer  //Oval image
   ovalID   as integer  //Oval sprite
   tweenID  as integer  //Sprite tween
endtype

global g as GameType

//*****************************************
//***          Main program            ***
//*****************************************
InitialiseScreen(1024,768,"Sprite Tweening", 0x686868, %1111)
LoadResources()
CreateInitialLayout()
do
   in = GetUserInput()
   HandleUserInput(in)
   UpdateTweenSprite(g.tweenID,g.ovalID,GetFrameTime())
   Sync()
loop

//*****************************************
//***           Functions              ***
//*****************************************
function LoadResources()
   g.imgOval = LoadImage("oval.png")
endfunction


function GetUserInput()
   result = GetRawKeyPressed(13)
endfunction result


function HandleUserInput(in as integer)
   global paused  //tween state 0:playing 1:paused
   //*** If Enter key pressed ***
   if in = 1
      //*** Change paused state ***
      paused = 1 - paused
      //*** Set tween according to paused state ***
      if paused = 1
         PauseTweenSprite(g.tweenID,g.ovalID)
      else
         ResumeTweenSprite(g.tweenID,g.ovalID)
      endif
   endif
endfunction


function CreateInitialLayout()
   //*** Create oval sprite ***
   g.ovalID = CreateSprite(g.imgOval)
   SetSpriteSize(g.ovalID,8,-1)
   SetSpritePosition(g.ovalID,10,50)
   //*** Create sprite tween ***
   g.tweenID = CreateTweenSprite(3.0)
   //*** Move sprite horizontally ***
   SetTweenSpriteX(g.tweenID,10,90,TweenSmooth1())      //*** Move
                                               ↳sprite vertically ***
   SetTweenSpriteY(g.tweenID,10,90,TweenSmooth1())      //***
                                      ↳Change sprite visibility ***
   SetTweenSpriteAlpha(g.tweenID,255,0,TweenSmooth1())
   //*** Rotate sprite by 720 degrees ***
   SetTweenSpriteAngle(g.tweenID,0,720,TweenSmooth1())
   //*** Double sprite size ***
   SetTweenSpriteSizeX(g.tweenID,8,16,TweenSmooth1())
   SetTweenSpriteSizeY(g.tweenID,GetSpriteWidth(g.ovalID),
   ↳GetSpriteWidth(g.ovalID)*2,TweenSmooth1())
   //***Change red setting
   SetTweenSpriteRed(g.tweenID,255,0,TweenSmooth1())
   //*** Play the tween ***
   PlayTweenSprite(g.tweenID,g.ovalID,3)
endfunction
```

Activity 1.7

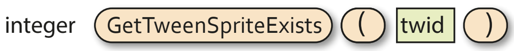
Modify *TweenSprite* to match the code given above and try pausing and resuming the tween by pressing the *Enter* key.

GetTweenSpriteExists()

To check if a sprite tween of a given ID exists, we have the `GetTweenSpriteExists()` command (see FIG-1.21).

FIG-1.21

GetTweenSpriteExists()



where:

twid is an integer value giving the ID to be checked.

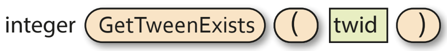
The function returns 1 if a sprite tween of the specified ID exists, otherwise zero is returned.

GetTweenExists()

To check if a tween (not only sprite tweens) of a given ID exists, we have the `GetTweenExists()` command (see FIG-1.22).

FIG-1.22

GetTweenExists()



where:

twid is an integer value giving the ID to be checked.

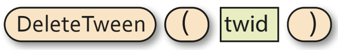
The function returns 1 if a tween of the specified ID exists, otherwise zero is returned.

DeleteTween()

If a tween is no longer used by any sprite (or other element) it can be deleted using `DeleteTween()` (see FIG-1.23).

FIG-1.23

DeleteTween()



where:

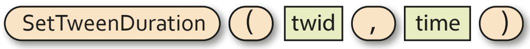
twid	is an integer value giving the ID of the tween to be deleted.

SetTweenDuration()

Although we normally set the duration of a tween when it is first created, we can change that duration later using `SetTweenDuration()` (see FIG-1.24).

FIG-1.24

SetTweenDuration()



where:

twid	is an integer value giving the ID of the tween manager.
time	is a real value giving the new time (in seconds) for the duration of the animation.

The duration of a tween should not be changed while it is playing.

Summary

- Tweening is the term used for the automatic generation of intermediate objects between given starting and finishing setups.
- AGK's sprite tweening commands are used to create tweens for sprites only.
- The commands allow a tween "manager" to store several behaviours.
- Behaviours allow characteristics such as the position, size, rotation, colour and transparency of an object to vary between its starting and finishing setups.
- To play through the generated sequence of a tween, a sprite is assigned to a specific manager.
- Use `CreateTweenSprite()` to create a sprite tween manager.
- Use `SetTweenSpriteX()` and `SetTweenSpriteY()` to add a change of position behaviour to a sprite tween. Positions of top-left corner of sprite.
- Use `SetTweenSpriteXByOffset()` and `SetTweenSpriteYByOffset()` to add a change of position behaviour to a sprite tween. Positions the
- Use `SetTweenSpriteAlpha()` to add a change of transparency behaviour to a sprite tween.
- Use `SetTweenSpriteAngle()` to add a change of angle behaviour to a sprite tween.
- Use `SetTweenSpriteRed()`, `SetTweenSpriteGreen()`, and `SetTweenSpriteBlue()` to add a change of colour behaviour to a sprite tween.
- Use `SetTweenSpriteSizeX()` and `SetTweenSpriteSizeY()` to add a change of size behaviour to a sprite tween.

- Use a method function call or an integer value when adding a behaviour to specify the "style" of the change.
- Use `PlayTweenSprite()` to link a sprite to a sprite tween manager and begin the tweening process.
- Use `UpdateTweenSprite()` each time an intermediate frame of a specific sprite tween is to be generated.
- Use `UpdateAllTweens()` to generate a new intermediate frame for every tween in the program.
- Use `GetTweenSpritePlaying()` to check if a sprite tween is currently running through its generated sequence.
- Use `StopTweenSprite()` to stop a sprite tween from "playing".
- Use `PauseTweenSprite()` to pause a currently playing sprite tween.
- Use `ResumeTweenSprite()` to resume a currently paused sprite tween.
- Use `DeleteTween()` to delete any tween manager.
- Use `GetTweenExists()` to check if a tween manager exists.
- Use `SetTweenDuration()` to change the duration of an animation.

Text and Character Tweening

Introduction

A similar set of tween commands to those we have just explored for sprites also exists for text resources and also for individual characters within a text resource.

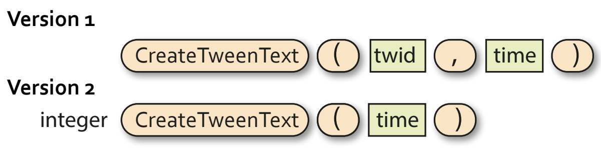
Text Tween Commands

CreateTweenText()

We can use `CreateTweenText()` to create a tween for linking to a text resource. The function has the format shown in FIG-1.25.

FIG-1.25

CreateTweenText()


Version 1
CreateTweenText (twid , time)
Version 2
integer CreateTweenText (time)

where:

twid	is an integer giving the ID to be assigned to the tween.
time	is a real number giving the nominal duration (in seconds) of the tween effect.

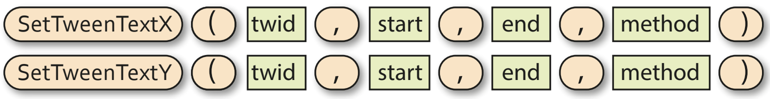
Version 1 of the command allows the user to select the ID assigned to the tween; version 2 assigns an ID automatically and returns the value assigned. Automatically assign IDs start at 100001. It is acceptable for tweens of different types to have the same ID.

SetTweenTextX() and SetTweenTextY()

These functions specify tween movement in the horizontal and vertical directions. The formats of the two functions are shown in FIG-1.26.

FIG-1.26

SetTweenTextX()
SetTweenTextY()


SetTweenTextX (twid , start , end , method)
SetTweenTextY (twid , start , end , method)

where:

twid	is an integer value giving the ID of the tween manager.
start, **end**	are real numbers giving the start and end *x* (or *y*) coordinate of the text.
method	is an integer giving the "style" of movement (these are described in FIG-1.5). A value of -1 stops all tweening.

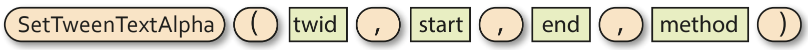
SetTweenTextAlpha()

To add a behaviour that adjusts the transparency of the text resource, we can use

`SetTweenTextAlpha()` (see FIG-1.27).

FIG-1.27

SetTweenTextAlpha()



where:

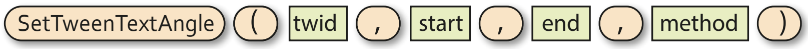
twid	is an integer value giving the ID of the tween manager.
start	is an integer number (0 to 255) giving the starting opacity. (0:invisible; 255: opaque)
end	is an integer number (0 to 255) giving the final opacity.
method	is an integer (-1 to 8) giving the "style" of change in opacity to be employed. The *method* options are comparable to those for movement.

SetTweenTextAngle()

We can have the tween rotate the associated text using `SetTweenTextAngle()` (see FIG-1.28).

FIG-1.28

SetTweenTextAngle()



where:

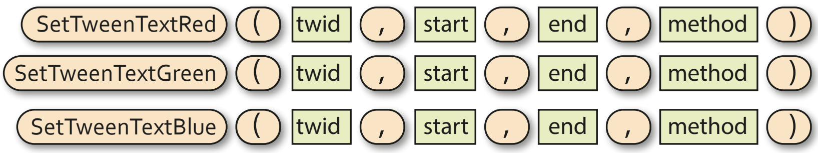
twid	is an integer value giving the ID of the tween manager.
start	is a real number giving the starting angle in degrees.
end	is a real number giving the final angle in degrees. This value can be greater than 360 if we want the text to perform several rotations.
method	is an integer (-1 to 8) giving the "style" of the change in speed of rotation

SetTweenTextRed(), SetTweenTextGreen() and SetTweenTextBlue()

The colour components of a text resource can also be tweened using `SetTweenTextRed()`, `SetTweenTextGreen()`, and `SetTweenTextBlue()` (see FIG-1.29).

FIG-1.29

SetTweenTextRed()
SetTweenTextGreen()
SetTweenTextBlue()



where:

twid	is an integer value giving the ID of the tween manager.

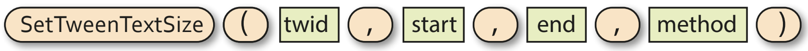
start	is an integer value (0 to 255) giving the strength of the specified colour at the start of the tween.
end	is an integer value (0 to 255) giving the strength of the specified colour at the end of the tween.
method	is an integer (-1 to 8) detailing the technique to be used in the colour change.

SetTweenTextSize()

We can tween the size of the text in a text resource using `SetTweenTextSize()` (see FIG-1.30).

FIG-1.30

SetTweenTextSize()



where:

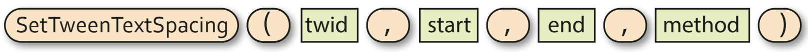
twid	is an integer value giving the ID of the tween manager.
start	is a real value giving the starting size of the text.
end	is a real value giving the final size of the text at completion of the tween.
method	is an integer (-1 to 8) giving the "style" of change in size to be employed.

SetTweenTextSpacing()

The space between the individual characters of the text resource can be tweened using `SetTweenTextSpacing()` which has the format shown in FIG-1.31.

FIG-1.31

SetTweenTextSpacing()



where:

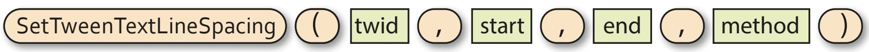
twid	is an integer value giving the ID of the tween manager.
start	is a real value giving the spacing between characters at the start of the tween.
end	is a real value giving the final spacing between characters at completion of the tween.
method	is an integer (-1 to 8) giving the "style" of change in size to be employed.

SetTweenTextLineSpacing()

Where a single text resource has text spanning two or more lines, the spacing between lines can be tweened using `SetTweenTextLineSpacing()` (see FIG-1.32).

FIG-1.32

SetTweenTextLine
↳Spacing()



where:

twid	is an integer value giving the ID of the tween manager.
start	is a real value giving the spacing between lines at the start of the tween.
end	is a real value giving the final spacing between lines at completion of the tween.
method	is an integer (-1 to 8) giving the "style" of change in size to be employed.

PlayTweenText()

To link a text resource to a text tween and have the effect "play" we need to use the `PlayTweenText()` function (see FIG-1.33).

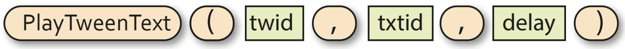


FIG-1.33

PlayTweenText()

where:

twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text resource to which the tween is to be applied.
delay	is a real number giving the delay (in seconds) before the tween should be applied.

UpdateTweenText()

To generate a new intermediate frame for a text tween, we can use `UpdateTweenText()` (see FIG-1.34).

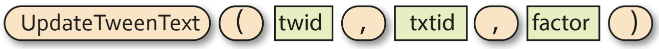


FIG-1.34

UpdateTweenText()

where:

twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text to which the tween is to be applied.
factor	is a real number affecting the speed at which the animation created by the tweening is played. Lower values mean a slower speed.

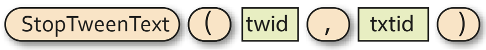
Remember, that although `UpdateTweenText()` is useful when only one or two tweens are running, where more are executing simultaneously, it is more efficient to use `UpdateAllTweens()`.

StopTweenText()

To stop a text tween that is currently playing, we can use `StopTweenText()` – the format of the command is shown in FIG-1.35.

FIG-1.35

StopTweenText()



where:

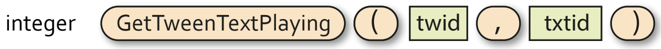
twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text resource used within the tween manager.

GetTweenTextPlaying()

To check if a text tween is currently playing, use `GetTweenTextPlaying()` (see FIG-1.36).

FIG-1.36

GetTweenTextPlaying()



where:

twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text resource used within the tween manager.

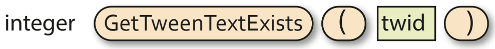
The function returns 1 if the tween is currently playing, otherwise zero is returned.

GetTweenTextExists()

To check if a text tween of a given ID exists, we have the `GetTweenTextExists()` command (see FIG-1.37).

FIG-1.37

GetTweenTextExists()



where:

twid	is an integer value giving the ID to be checked.

The function returns 1 if a text tween of the specified ID exists, otherwise zero is returned.

The program below creates a two-line text resource which makes use of a text tween to arrive at its final position, size, colour, and orientation.

```
// Project: TweenText
// Created: 2016-07-08

//*** Include library files ***
#include "../Function Library/CoreLibrary.agc"

type GameType
   textID   as integer     //ID of text resource
   txtTweenID  as integer  //ID of tween resource
endtype

global g as GameType
//***************************************
//***         Main program            ***
//***************************************
InitialiseScreen(800,800,"Text Tweening", 0xA8A8A8, %1111)
CreateInitialLayout()
do
   UpdateTweenText(g.txtTweenID,g.textID,GetFrameTime())
   Sync()
loop

//***************************************
//***             Functions           ***
//***************************************
function CreateInitialLayout()
   //*** Set up text resources ***
   g.textID = CreateText('Badges? Badges?\nWe don\'t need no
   ↳stinkin\' badges!')
   SetTextColor(g.textID,0,0,0,255)
   //*** Create the text tween ***
   g.txtTweenID = CreateTweenText(3.0)
   //*** Add tween behaviours ***
   SetTweenTextX(g.txtTweenID,50,10,1)
   SetTweenTextY(g.txtTweenID,10,50,1)
   SetTweenTextSize(g.txtTweenID,0.3,4,1)
   SetTweenTextBlue(g.txtTweenID,0,255,6)
   SetTweenTextAlpha(g.txtTweenID,0,255,2)
   SetTweenTextAngle(g.txtTweenID,-90,0,1)
   //*** Play text tween ***
   PlayTweenText(g.txtTweenID,g.textID,0.0)
endfunction
```

Activity 1.8

Start a new project called *TweenText* and implement the code given above.

PauseTweenText()

To pause a currently executing text tween, we can use `PauseTweenText()` (see FIG-1.38).

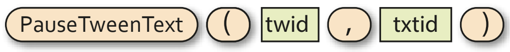


FIG-1.38

PauseTweenText()

where:

twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text resource used within the tween manager.

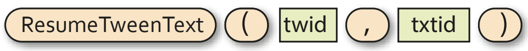
ResumeTweenText()

To resume a currently paused text tween, we can use `ResumeTweenText()` (see FIG-

1.39).

FIG-1.39

ResumeTweenSprite()



where:

twid	is an integer value giving the ID of the tween manager.
txtid	is an integer value giving the ID of the text resource used within the tween manager.

Character Tween Command Set

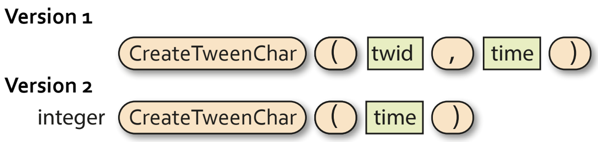
The individual characters within a text resource can be assigned their own tween. This allows us to have individual characters perform their own animation independently of any animation that the parent text is performing. For example, the text resource might be moving from the top-left to the bottom right of the screen while the first character in that text is changing colour from blue to red.

CreateTweenChar()

Creates a tween for linking to a character within a text resource. The function has the format shown in FIG-1.40.

FIG-1.40

CreateTweenChar()



where:

twid	is an integer giving the ID to be assigned to the tween.
time	is a real number giving the nominal duration (in seconds) of the tween effect.

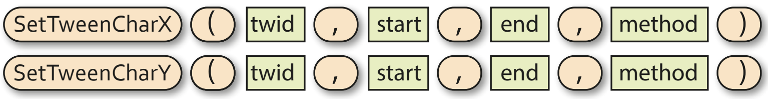
Version 1 of the command allows the user to select the ID assigned to the tween; version 2 assigns an ID automatically and returns the value assigned. Automatically assigned IDs start at 100001. It is not acceptable for tweens of different types to have the same ID.

SetTweenCharX() and SetTweenCharY()

These functions specify tween movement in the horizontal and vertical directions. The formats of the two functions are shown in FIG-1.41.

FIG-1.41

SetTweenCharX()
SetTweenCharY()



where:

twid	is an integer value giving the ID of the tween manager.
start, **end**	are real numbers giving the start and end *x* (or *y*)