

AngelScript Basics

3DRAD

3D engine

```
// initialise the obstacles Array as a Global Array
// remember a Global Array or variable can
// be read from all part of your script.
string filename;
string report = "";
int er;
Index px={0,0};
int [] botx={0,0}; // bot x nodes
int [] botz={0,0}; // bot z nodes
int [] botc={0,0}; // bot counters
int [] botSkinMesh={OBJ_89,OBJ_44}; // bot SkinMesh
int [] botAndroSkinMesh={OBJ_735,OBJ_917}; // bot Andro SkinMesh
Vector3 xyzPathfinder;
int AnimationSet = 0;
int AnimationSetNew = 0;
int []turn={0,0};
void Main()
{
    if(!initializing()){ //initializing
        //memorize initial character
        string filename="walkableData.txt";
        bots=loadFile(filename);
        player=0;
    }
    //end initializing

    //main loop
    switch( dst ){
        case 1: if(IN_939<botstart&&turn[0]==0){ // event on pr
            turn[0]=1;
            player=0;
            bots=initialiseBot(player);
        }
        dst=2;
        break;
        case 2: if(TW_939>botstop){

```

By Darryl Hunt

AngelScript is the 3D RAD's script interpreter, some have described it similar to C and in many ways it is but it has some differences. AngelScript is a programming language that can be used to access many of the 3D RAD functions. A description of these functions can be found in the [Script_Object_Reference.rar](#) download above. AngelScript has most of the features found in hi-level programming languages and with the added 3DRAD functions it makes it a very comprehensive language for 3D programming.

In this section I will describe what reminder statements are and the format of the Main() function. Also we will be taking looking at what a variable is and how it works for you.

Next I must explain the reason for the 2 forward slashes followed by a description. These are the reminder statements, when you write your code it helps to put reminder statements every now and then so when you look at it in the future it will help you remember how it works. I have included more reminder (rem for short) statements in the following code sample. These rem statements describe the operation of each section of this example code.

The first line of code is the reminder statement telling you what this code is for. Second we initialise the count variable (int count = 0;) This defines integer variable called 'count' and sets its value to zero. When declared before the Main() function it only happens once and any variable you plan to use throughout your code must be defined here. This is called a global declaration, which means the variable can be read by any function you create. Optionally you don't have to set its value (eg. int count;), or it can be set to any value like (eg. int count = 5;).

An integer can have a value from **-2,147,483,648 to 2,147,483,647** * a float point number value is **-/ +3.402823466E38**. Variable names can be almost any name consisting of letters or numbers in most cases you can not use any character values like the following `{ } [ ] < > , / ? \ | ) ( & ^ % $ # @ ! " ' ~ ` = + - . : ; ' ,` or a space. Angelscript has some reserved words you can't use it would be best to check out their website for details. The best naming method for a variable is to use a name that represents what the value might be used for, referring to the 'count' variable used in the sample code.

1

This part of code (void Main(){ code }) is the primary function and holds the main body of your code between the curly brackets. The curly brackets are important and define the start and end of your code. 3D Rad loops through here 75 times per second. Void Main() the name 'Main' is reserved for Angelscript as the primary location for your script and can not be changed or used elsewhere in your code. Functions can return a variable containing a value but in this situation there is no point because there is nothing to do with it, which brings us to the 'void' declaration before the 'Main'. The word 'void' declares that the data type the function is going to return, in this case mean none.

You can also define functions of your own that return values but I won't get into that now.

Void is not really a data type it is used to tell the compiler that a function doesn't return any data.

Next in the script on the previous page you see here 2 if() statement these are decision statements. You can see from these 2 if() statement samples there is a difference in the way they are laid out, the second one has curly brackets encompassing the statements. The curly brackets are not totally necessary but help you understand where your statements start and finish. For programming convention of newbies its best to use curly brackets for all (eg. if(){ // code }) statements. For the record the if() statement doesn't need curly brackets in some circumstances. The format must be all on one line like this sample you can even have more than one statement on a line.

```
if (count < 200) OUT_0 = iFloatRand(0,1);
```

You can stack other if() statements inside another but more of that later. Every entry of code you create must end with a semicolon (;) this tells the compiler that this is the end of the code so it moves to the next entry.

In the script you see the equal sign twice and also by itself. The 2 equal signs are used to compare variables and the single is used to assign a value to a variable. The 2 equal signs are called comparators. Using these inside an if() statement comparators can check what the value of a variable is. There are many types of comparators there is more about comparators and decision-making in the next chapter.

The last line of code (count++) increases the value in the 'count' variable by 1. This can also be written as (count=count+1) but this is little more complex and really not necessary. If you want to 'count' more than 1 then use this method example (count+=5) or (count=count+5).

Declaration of Variables: here is how you declare variables;

```
int aa; // this declares an integer variable called aa that has no value
int aa=10; // this declares an integer variable called aa with the value of 10
int aa,bb,cc; // this declares 3 integer variable called aa bb and cc that has no value
int aa-1,bb=2,cc=3 // this declares 3 integer variable called aa bb and cc that have been assigned values
```

The same format can be used for Float and String variables. String variable value declaration must have Quote marks like the following

```
string aa="Hello World";
```

Declaration of arrays is covered in the Arrays chapter. There are 2 more variables to consider Vector3 and Quaternion. These variables are used by 3DRAD math functions. Declaring them is done the same as other variables.

```
Vector3 directionzyx;
Quaternion rotationzyx;
```

Last for the un-initiated you don't have to declare variables your not going to use. Sometimes when developing a script you might add variables that end up excess to your requirements and even for debugging purposes. So when your finished developing a script go over it and remove excess variables, debug variables and don't forget to add rem statements so when you look at your code in the future you will know how it works.

String variable is a variable that's value is a character or a number of characters. Like other variables you must declare a variable before you assign anything to it. For example;

```
string text="a";
string text="abcdefghijklmnopqrstuvwxyz";
```

It will also represent all the characters on the keyboard.

3DRAD now has many functions to help manipulate strings but there are few basic things you can use in AngelScript.

First is adding strings together

```
text="John "; name="Thomas";
text+=name;
```

This results is "John Thomas", notice I added a space at the end of John so the resulting string would have a space between them.

Next you can add a numerical variable to the end of a string

```
int score=10;
text="Score "+score;
```

This results in "Score 10"

Things you can't do is with other math functions are multiply, divide and minus strings.

There are some special characters you can use to format the display of a string.

"\r" is a carriage return or new line. This character is not displayed but moves the text after it to the next line.

```
Text="This is a test to \rsee how it works!";
```

It would be displayed as follows

```
"This is a test to
see how it works!"
```

From my experience with 3DRAD Text print object the following doesn't work. Text=score; where score is an integer or float variable.

You must do the following text="" +score;

Designing efficient code

Above I mentioned you should design your code more efficiently and don't over use looping statements. If you are new to coding I am about to mention some code statements and processes you may not be familiar with but these are explained in later chapters.

The main rule of programming is the K.I.S.S. rule Keep It Simple Stupid. There are in most cases more than one way to do the same job. So after you have designed a piece code don't sit back and say well it works I better not mess with it. There is often something that can be done better, simpler and don't forget to include rem statements. Loop statements are for checking, comparing all kinds of information but these will stop the frame rates if the process is large. Like I said before there is always another way to do the same job without slowing things to much. Below are 2 sample methods for checking the same list of data. Say we want to check a list of player scores in an array. The following for() loop statement will do this fine but this code will slow frame rates compared to the next sample.

This for() loop code example will not slow the process very much on its own but if where duplicated one for each player or had to count through a larger array it will slow frame rates.

```
int []player = {0,0,0,0,0,0} // player score array
void Main()
{
for ( int l = 0; l < 5; l++ ){ // start loop
    if ( player[l] > 0 ){ // check if player is initialised
        player[l]++; // if so then increase score
    }
}
}
```

This code is processed 75 times a second and incrementing the counter each pass through. This checks 1 array element each pass through then the counter value is incremented to check the next array element for the next pass through. This is a much more efficient way of checking repetitive data.

```
int []player = {0,0,0,0,0,0} // player score array as a global array
int l=0; // initialise as a global variable
void Main()
{
    if ( player[l] > 0 ){ // check if player is initialised
```

```

    player[l]++; // if so then increase score
    if (l==5){
        l=0;
    }else{
        l++;
    }
}

```

Another efficient pass through method is the switch() statement. This can be incremented like the last sample but have multiple if() statements to check like the following example. Each statement is checked in turn once every 1/75th of a second. Each process following the case statement can have more than one statement and don't have to be identical processes like in our sample. There is more about the switch() statement in a later chapter.

```

int player1=0;
int player2=0;
int l=0; // initialise as a global variable
void Main()
{
    switch(l){ // switch is incremented every pass through
        case 1:
            if ( player1 > 0 ){ // check if player is initialised
                player1++; // if so then increase score
            }
            break;
        case 2:
            if ( player2 > 0 ){ // check if player is initialised
                player2++; // if so then increase score
            }
    }
    if (l==2){
        l=0;
    }else{
        l++;
    }
}

```

There are processes that should never be done during game play like loading or saving data these processes must be done at the start or end of the game. If you have to position players you load the data, then at the start of the game you initialise them don't do this during game play. If it's necessary then load the data into an array because these are memory based they work much faster than load and save statements.

Debugging your code: First 3DRAD's Script editor has an excellent debug button use it regular.

Next if things don't go as planned during the game then print the variables in question to the screen so you can see what's happening. To do this add a iObjectTextSet(OBJ_x,string); to your code. Place it at the bottom of the code outside any loops or if() statements. Then declare a global string variable called debug.

```
debug="";
```

Add the debug variable to the code next to the other variables you want to check like this.

```

If(damage>0){
    score++;
}
debug="score "+score+" Damage "+damage;

```

This will display the score and the damage variables on the screen during game play and you can see what's happening. You can use this method for vectors, integers etc.

Setting out code: You may have notice from the code samples so far how some lines are indented from others. This helps to identify code blocks like if() statements, loops and even functions. This is not mandatory its just to help you to visually identify blocks of code and what they might contain. A block of code might contain other blocks of code. Imagine or even try aligning all your code to the left and you will see how much more difficult it is to find blocks of code. My preference is to have the function aligned left then the first declaration or code block as the first indent. All the code within the block as the next indent. When that block ends then the next block is at that same alignment. Check the code example on page 1.

Notes

Decision making.

This discussion is about decision making and covers the if() statements and comparators in a little more detail. First lets start with the If() statement.

If() statement has many forms below are some.

In the first sample if the comparison is TRUE then the code between the curly brackets is processed. If the comparison is FALSE then it processes the next line of code after the last curly bracket. You have most likely seen or used this format all ready.

```
If(comparison){
    // do something here
}
```

In this next if(){else} combination, the comparison is TRUE then the code between the first set of curly brackets is processed. If it is FALSE then the code between the second set of curly brackets is processed. You could call this a flip-flop system where one process is done or the other.

```
If(comparison){
    // do something here
}else{
    // else do something here
}
```

Now this if() statement is a little more complex. If the first comparison is TRUE in this next if() statement then the code between the first set of curly brackets is processed. It then skips the next comparison in the second argument and processes the line of code after the last curly bracket. If the first comparison is FALSE then it checks the next comparison for TRUE or FALSE. If this comparison is TRUE it processes the code between the second set of curly brackets. If it is FALSE then it skips to the next line of code following the last curly bracket. This has a similar operation to 2 sets if() statements, one after the other like the first sample, except if the first comparison is TRUE it skips the second comparison.

```
If(comparison){
    // do something here
}else if(comparison){
    // else do something here
}
```

The following set of statements are call stacked if() statements. The first if() does a comparison and if TRUE it processes the next if() statement. If the comparison is FALSE then it skips the internal if() and continues the code after the last curly bracket. This is only useful if there is something else you want to process as well as check the internal if() statement. If you have two conditions to compare then it can be done with logic comparators in one statement shown in the next sample.

```
if(comparison){
    // do something
    if(comparison){
        // do something
    }
}
```

This is called a logic AND comparison. If you have 2 sets of conditions to compare before you process something then this is how you do it. If the first comparison is TRUE AND the next is TRUE then the code between the curly brackets are processed.

```
if(comparison && comparison){
    // do something
}
```

In all the code samples above I have used the word 'comparison' instead of a comparator algorithms (eg. value1 <= value2). Click the link above and browse the various comparators and don't be afraid to experiment. Most are just what they look like

== equal to
<= less than or equal to

>= greater than or equal to
> greater than
< less than
!= NOT equal
&& Logic AND
|| Logic OR

These are the most popular ones you will use. The comparison precedence is always done where the LEFT is compared to the RIGHT. So if the LEFT value1 is less than or equal to the RIGHT value2 then the result is TRUE (eg. value1 <= value2).

The following script uses a stacked if() and flip-flop method to turn on or off sound or it could be a random light flashing like thunder or lightning. To a new project add a sound effect object and a script object. Paste the code below into the script text editor and edit the OBJ_X to whatever the sound object value is. Exit the script and press space bar to run this code. You should hear random sound of varied length. To alter the length of the sound increase or decrease the number 3 in the iFloatRand() function.

```
int sw = 1;    // define sw variable and set to 1
int randEvent; // define randEvent variable
void Main(){
  randEvent = iFloatRand(1,3); // random number between 1 and 3
  if(randEvent == 1){ // this waits for a random number of 1
    if(sw == 1){ // this checks if the sound is on
      sw = 0;    // this sets sw variable to zero
      // turn off sound
      iObjectStop(OBJ_X);
    }else{      // otherwise it turns the sound on
      sw = 1;    // this sets the sw variable to one
      // turn on sound
      iObjectStart(OBJ_X);
    }
  }
}
```

Notes

Functions.

In the world of scripting it can be very complicated and to the inexperienced very confusing. To reduce confusion it helps to break down your code into segments. If you have any code that's repeated over and over, this also makes scripts excessively large. This where the function comes in, a function holds code that can be accessed many times through out your script. Say for example you want to keep track of a player's position from two or three different areas of your script then instead of writing the code many times you just call the one function from those places. This function would contain the code that does the comparison and returns the result back the main script. This not only simplifies your main body of code it helps you debug any problems by isolating sections of code. Another benefit is you can reuse functions in other scripts without having to rewrite and debug it.

A function doesn't have to return a value but in most cases you will have it calculating something or searching for a value so you will want to return a value. To do this you must declare the nature of the value your function will return like an integer, floating point number etc.

To call a function that returns a value you simply assign it to a variable like ' eg. `play=myGameOver(damage);` '. In this example the function receives a value in the variable name 'damage' then does something with it in the `myGameOver()` function and returns the result to the 'play' variable. If a function doesn't return a value then you call it like this '`myGameOver()`' you can pass it a value like the previous example if you want. Functions can receive more than one variable but you must separate them with a comma ',' like this example `play=myGameOver(damage, life, shield)`.

The example below shows how you declare a function that will return an integer value. This is a very simple function you pass it a value which is received as the integer variable in `aa` and in the following line `aa` is increased by 1 and returned to the calling argument.

```
int myFunction(int aa){  
  
    return aa++;  
  
}
```

This function receives multiply values.

```
int myFunction(int aa , int bb){  
  
    return aa + bb;  
  
}
```

For a function to accept a variable it must be declared in the function header declaration like the samples above. If you want another variable inside the function you declare it as in the example below. This variable is now only readable from within this function and can even have the same name as other variables in your code except global variables. When the processor exits this function the variables value is lost so all the variables in the following example are only readable from within the function. I have had a situation in the past where you want to preserve a value in the function but you can't so you just find another way to do it. The function can only return one value this must be the same value (int – float – string) as in the function header declaration.

```
int myFunction(int aa , int bb){  
    int cc=10;  
    return aa + bb+cc;  
  
}
```

Function name can be almost anything but there are some reserved words the most obvious is the `Main()` word used by AngelScript as the primary function. For details check the AngelScript website. 3DRAD also has function names you can't use. The best method for names is to use a name that relates to the functions purpose for example if it the function is to keep track of a players life use a name reflecting that. You can use any letter or number but not any character like '`{ } [ ] < > , / ? \ | ( ) & ^ % $ # @ ! " ' ~ ` = + - . ;`' or a space. AngelScript is case sensitive and you must use the same case to call a function as what you declared it with. This means that `myGameOver()` is not the same as `myGameover()` this goes for all functions including the 3DRAD inbuilt functions..

Below is an example of how you might layout a function in the 3D RAD editor. This script calls my user-defined function called `myDamageCounter()` from two different positions. It sends different values to the function depending on the damage you may want inflict on the player, and where they may have been struck, more for a head blow, less for a body blow. This function will return an integer value based on the results of the code inside the function. I have also defined 3 internal integers, these integers can only be accessed and changed from within this function. If you

read through the codes rem descriptions you will see how it works. If removed the function and included the lines of code the main script, twice you could see how much more complex it would become.

```
int count=0; // declare count variable
int headBlow=0; // declare headBlow as global variable
int bodyBlow=0; // declare bodyBlow as global variable
void Main(){ // AngelScript main function
    headBlow=IN_44; // get a trigger value from the Value object
    bodyBlow=IN_45; // get a trigger value from the Value object
    if(headBlow > 0){ // if headBlow is greater than zero
        count= myDamageCounter (count, 10); // increase by 10 (more damage than head blow)
        OUT_44=0; // zero trigger in Value object
    }
    if(bodyBlow > 0){ // if bodyBlow is greater than zero
        count= myDamageCounter (count, 5); // increase by 5 (less damage than body blow)
        OUT_45=0; // zero trigger in Value object
    }
    if(count== -1){ // if count equals minus one then end game
        iObjectStart(OBJ_X) // end game with exit fade object
    }
}

int myDamageCounter(int aa ,int bb){ // user defined function
    int cc=aa+bb; // declare an internal variable and assign new value
    if(cc >= 100){ // check if greater than 100
        return -1; // return minus 1 to end game
    }else{
        return cc; // return increased damage count by cc
    }
}
```

This sample function increases the count value by five or ten depending where it was called from or minus one if the count is equal to or above one hundred thus ending the game.

I not sure this sample is all that useful but it demonstrates how functions work, how you can reduce code and simplifies it overall, making it easier to debug. 3D RAD has its own built-in functions, which can be used from the Script object. Your user-defined functions basically follow the same conventions as 3D Rad defined functions. So what you have learnt so far calling these 3D RAD functions is the same.

Notes

Looping

AngelScript has 3 looping statements these can be used for a variety of reasons. The following is the correct syntax for these arguments. You don't have to use the curly brackets but if you do you must have the looping statement and only one following statement on the same line. For learners it is best to use the curly brackets because they help identify where the start and end of the arguments are. All these loop statements process the code between the curly brackets until the condition in the `BOOL_EXP` is true.

```
while( BOOL_EXP ){  
    STATEMENT  
}
```

```
do{  
    STATEMENT  
}while( BOOL_EXP );
```

```
for( INIT ; BOOL_EXP ; NEXT ) {  
    STATEMENT  
}
```

The following function contains a 'while' loop. This loop keeps processing the code inside this loop while the variable 'cc' is greater than zero. The boolean expression is evaluated before processing the code between the curly brackets and if true it then continues processing the next line of code in this case the 'return' statement. The fundamental difference between the 'while' and the 'do while' loops is that the 'while' loop evaluates the expression before execution the code and the 'do while' loop processes the code before it has evaluates the expression.

```
float _pow(float num, float exp){ // num to the power of exp  
    int cc=exp;  
    float ret=1;  
    while(cc>0){  
        ret*=num;  
        cc--;  
    }  
    return ret;  
}
```

So in the next example I have used the 'do while' loop to show the difference. Now with the boolean expression following the code the variable 'cc' will decrease one more to -1 and the 'ret' variable is multiplied extra time before the expression is evaluated. This would send the wrong value to the 'return' statement and the function would return the incorrect amount. To correct this we would change the expression to read (`cc>1`). This demonstrates the difference between these two statements.

```
float _pow(float num, float exp){ // num to the power of exp  
    int cc=exp;  
    float ret=1;  
    do{  
        ret*=num;  
        cc--;  
    } while(cc>0);  
    return ret;  
}
```

The 'for' loop is a counting loop. In the next example I have substituted the 'for' loop and made some other changes. In the first part of the 'for' expression we set up a variable. You can assign the value from another variable as in the example below or like this (`cc=10;`). If the variable has not been initialised before you must do so (`int cc=10`). The boolean expression can use any comparators you may have used in the 'while loops or even an 'if' statement. There are many types of comparators for more on comparison operators go to the AngelScript web site. The next part of the 'for' statement you can increase or decrease the counting value. For more about these operators go to the AngelScript web site. If want to count by more than one you can, you just count do this (`cc+=5`) this increases the counter by 5. You can just use it as a counting loop like the example or you can use the counting variable to increment some other variable with in the loop.

```
float _pow(float num, float exp){ // num to the power of exp  
    float ret=1;  
    for (int cc=exp; cc>0; cc--){  
        ret*=num;  
    }  
}
```

```
        return ret;  
    }
```

You can also stack loops, that is put one inside another. You may want to evaluate an x/y coordinate where the outer loop is for x and the inner loop is to evaluate the y coordinate. If you want to end a loop prematurely you use the 'break' statement. This will only end the loop that contains it; it will not end an outer loop if it's contained within the inner loop.

Notes

Arrays and Array handling

An Array is an orderly arrangement of elements that can contain different values all using a common name. Arrays are very commonly used to handle list of data like file information.

Bear with me for this next description as it's for those who have never come in to contact with Arrays. An Array is like a row of boxes (Array elements) lined up so you can look in them one at a time to see what each might contain or you may want to put something in them. Now if you are handling numerical values, each box (Array element) can contain only one value. You can use an individual box (Array element) in a math formula but more on this later. To look through a line of boxes (Array elements) you use a loop statement, these I covered in the last chapter I will also demonstrate how you use them with Arrays. I mentioned above an Array has a common name for all these elements. The naming of an Array follows what I have previously described about naming variables in an earlier chapter on variables. An array can be numerical or string Arrays. Like normal variables you must declare an Array. AngelScript uses three types of values the Float point Number, Integer and String. To declare an Array you do this.

```
int[ ] a ;
```

This declares an Integer Array called 'a'.

```
float [ ] b;
```

This declares a Floating point Number Array called 'b'. Now these Arrays have only one element with nothing in it. If you want an Array with more elements you declare is like this.

```
int[ ] a (10) ;
```

This initialises an Array called 'a' containing 10 elements with nothing in them. You can also initialise an element with a value shown next.

```
int[ ] a = { 1,10,4};
```

This initialises an Array called 'a' with three elements the first element value is one the second is ten and the third is four.

This is the basics of how you declare an Array next we will show some uses for an Array but first a word on how an Array is indexed.

An array element numbering convention begins with the first element and this is zero the next element is 1 the next is two and so on. So the first element is number zero element and not number one element. Confused well you wouldn't be the first, but to handle Arrays you must get a grasp of this concept. For example you may have a scoring system that has player 1, 2, 3 and so on. To represent this in the Array it would be

```
score[0] = 1;  
score[1] = 2;  
score[2] = 3;
```

If you where to check the Hi Scores for 3 players the following for loop gives us an example of what might need to do. This loop counts from 1 to 3 the if() statement checks the 'newscore' against the score in the Array. If the 'newscore' is greater than the old score stored in the score Array it will assign it to the 'HiScore variable. If its not higher it will then continue comparing the next Array element. The other significant thing is that we used minus 1 in the Array argument because the elements start at zero.

```
for(count=1; count<=3; count++){  
    if(newscore> score[count-1]){  
        HiScore=newscore;  
    }  
}
```

AngelScript Arrays can be only single dimensioned unlike C which can have multidimensional array. For information purposes a multidimensional array has the following syntax.

```
float [ ][ ] cords (10,10); // this is a floating point 2 dimensional array with 11 elements in each dimension
```

You will only get an error when you declare an array like this it is not possible to have multidimensional arrays in AngelScript. If you are converting code from C or other languages then you will have to find another way to do it.

For our example we will develop a file read function. The read function will require a filename and will load an Array called 'score' containing the file data.

You are not able to pass an Array to a function or return an Array from a function so you must declare it as a Global Array this makes it accessible any where in your script.

```
// initialise the 'score' Array as a Global Array
// remember a Global Array or variable can
// be read from all part of your script.
int[] score (5);

void Main()
{
    //Your script goes here...
}

void SaveMyScore(string filename){           // write file function
int filehandle= iFileWriteOpen(filename);    // initialise the filehandler variable
int count=0;                                // initialise the count variable
int arrayLen=score.length( );               // get the array length from the Array length method
while(count< arrayLen){                     // while loop counts from 0 to array length
    iFileValueWrite(filehandle, score[count], true); // save each value from score array to the file
    count++;                                // increase the count
}
iFileClose(filehandle);                     // clean up by closing the file
}                                           // exit function
```

This function accepts a 'filename' as a 'string' from the calling argument, it then calls the 3Dread function to get the file handle. Next it initialises the 'count' variable then it get the array length from the Array length method. This variable gets its value from the 'length()' method, which is a method of the Array. Arrays have 2 methods 'length()' and 'resize()' you can use the 'length()' method as we have done to find the number of elements it contains. The 'length()' method returns number of elements so an Array that contains 10 elements that would number from 0 to 9. The 'resize()' method is used to increase or reduce the number of elements in an Array but more on this later. Now back to our function, I used a 'while' loop to retrieve all the data from the 3Dread FileValueRead() function. This 3Dread function only retrieves one value at a time so this value is assigned to the 'score[]' Array. The 'count' variable indexes the Array ready for the next value from the FileValueRead() function. When the 'count' variable is greater than the 'arrayLen' there no more values from the file so the comparison in the 'while' argument will be true ending the loop. Last we clean up by closing the file and returning to the main script.

With this function we can write to a file all we have to do is pass it a filename.

```
void GetMyScore(string filename){           // read score from file
int filehandle= iFileReadOpen(filename);    // get file handle
int count=0;                                // initialise count variable
int arrayLen=score.length( );               // get array length from the Array method
while(count< arrayLen){                     // while loops from count to array length
    score[count]=iFileValueRead(filehandle); // load score array from file
    count++;                                // increase count
}
iFileClose(filehandle);                     // clean up by closing file
}                                           // exit function
```

First the function accepts a 'filename' from the calling script. First it gets the 'filehandle' variable then initialises the 'count' variable. Next it initialises the 'arrayLen' variable and get its value from the 'length()' method. To get the data from each element in the Array we use the 'while' loop this continues to loop until the 'count' variable is greater than the 'arrayLen'. The 3Dread FileValueWrite() function writes the value from each element in turn until the loop comparison is true. Last we have to close the file and the function ends.

Counting elements that don't exist may cause an error. If you had an Array that contained 10 elements and a loop that counts to 10 it may cause an error because the indexer has exceeded the number of elements so we use the length() method to find the number of elements you may need to subtract one before you start counting this can depend on the loop statement you have used or the way you have used it.

You can assign a value to any element in an array as follows.

```
a[0] = 1;
a[5] = 10;
a[8] = playerone;
```

This assigns the value of 1 to zero element, 10 to element 5 or value from 'playerone' variable to element 8. You can use an Array element in a formula like the next sample.

```
c = a[5] - a[8];
```

This assigns 'c' with the value of the difference between element 5 and element 8.

You can increase the size of an Array by assigning new elements with the 'resize()' method. If you had an Array that had 10 element you can increase it to 15 elements like so.

```
score.resize(15);
```

This Array will now contain 15 elements numbering from 0 to 14. One more thing you must resize an Array before you get its number of elements with the 'length()' method or it will cause an error.

You may think I have covered all there is to know about arrays but this is only the tip. Arrays have many more uses and many more tricks but above all they are extremely useful. This instalment should give you enough information to get started with using Arrays in conjunction 3Drad file functions.

Notes

Switch / Case

The following script sample is the correct syntax for switch/case. The INT_EXP is the integer value you want to evaluate. INT_CONST is the value that initialises the TRUE condition. Each case block should end with a break statement unless you want to continue to evaluate the value in the following case statements or process the default statement.

```
switch( INT_EXP )
{
case INT_CONST:
    STATEMENT

case INT_CONST:
    STATEMENT

default:
    STATEMENT
}
```

The default statement is optional. The following code evaluates the randEvent number that may be 1 or 2. If 1 then the first case statement will be TRUE and the code after that case will be processed.

```
switch( randEvent )
{
case 1:
    // do something
    break;

case 2:
    // do something
    break;
}
```

Each case argument can have multiply statements like if(), Loops etc. You can have many case arguments and used correctly it is a very efficient method of controlling code flow.

Notes

