

Games Programming Club Making an 'Object Avoiding Game'

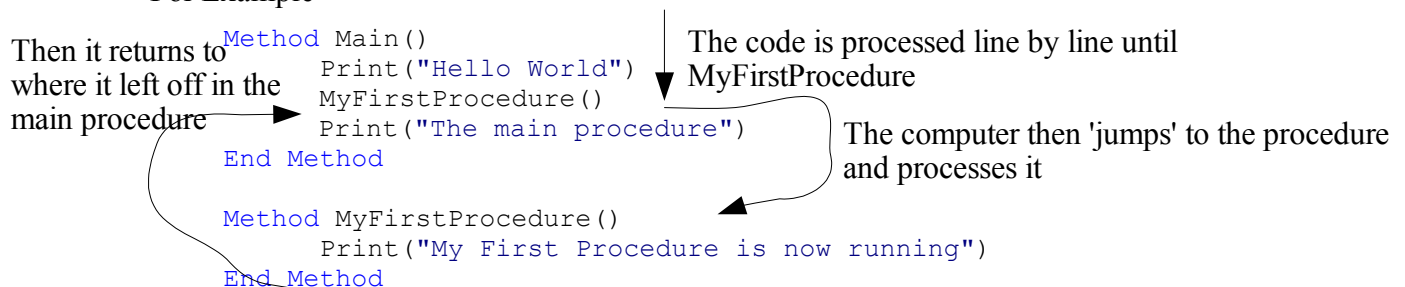
We're going to be making a game, which is very similar to the built in game 'Planes'. The game will require the user to 'drive' their UFO around the screen, so that they avoid the baddies.

If you get stuck, there is also an accompanying solution booklet to help, but try to do it from what you know, as far as possible.

Firstly, start a new program

1. A procedure is like a 'subprogram'. When your program runs, the computer goes to 'method Main()' and then goes down each line of code, one by one until it gets to 'End Method'. However, what a procedure does is that when the computer gets to the name of the procedure in your code, it will 'jump' to another part of your program and then when it has finished with that it will jump back.

For Example



Use the construction below to create a procedure called 'LoadMainSprite'. Also use the 'End Method' code to show the end of your procedure.

```
Method NameOfMethod()
```

2. In the Method Main() procedure, we want to "call" the 'LoadMainSprite' procedure, which is a fancy way of saying that we want to run the procedure. This is done by writing the name of the procedure you want to call followed by ().

```
NameOfMethod()
```

3. Now we want to write the code inside our procedure – after all there is no using jumping to another part of the program, if we just jump back again. Write the code for the procedure 'LoadMainSprite', which should load the UFO (UFO.gif) and call it "MainSprite". The Sprite should then be moved to location 20, 200 on the screen and it should be shown

```
LoadSprite("Name of sprite", "File to be loaded")
MoveSpriteToPoint("Name of sprite", location, location)
ShowSprite("Name of sprite")
```

4. Test your program is working. Now we are going to make the UFO look like it is spinning, in order to do so, copy the following code at the end of your LoadMainSprite method.

```
Timeline[1] = 100
Timeline[2] = 100
Timeline[3] = 100
Timeline[4] = 100
Timeline[5] = 100
Timeline[6] = 100
```

```
SetSpriteAnimationTimeline( "MainSprite", True, Timeline )
```

- Now we are going to set the background as a picture of space. To do this, use the same technique as you did in steps 1 and 2 to make a method called 'DrawBackground'
- Next, we are going to write the code to go inside the DrawBackground procedure. Firstly use the same technique as in step 3 to load a sprite whose name is 'Background' and the file to be loaded is 'Space.png'
- Type the following code at the end of your DrawBackground procedure, which will set Background as the background and stop our program detecting collisions with it.

```
SetSpriteCanCollide("Background", False)
StampSprite("Background")
```

- Then, using the same technique as in steps 1, and 2, create and call a procedure called 'CheckForAnArrowKey'.
- Now we have a sprite showing, however we want to be able to move it around with the arrow keys. In order to do this, we need to constantly ask 'is the left arrow being pushed', 'is the right arrow being pushed' etc. This is done by the following code

```
If IsKeyDown("Direction") Then
    //More code will be added here later
End If
```

Using this, write, inside the 'CheckForAnArrowKey' procedure, 4 of the above functions, (one for each direction; left, right, up and down), one after each other.

- At the moment our code looks to see if any direction arrow is being pushed, however if a key is being pushed, our program does nothing. We will now 'handle the events', to make our program react to key presses. Firstly, create 2 global variables (of type Int), called XCoordinateofMainSprite and YCoordinateofMainSprite.

To do this, write at the very top of the program, (but below [Program Untitled1](#)) the following

```
Define XCoordinateofMainSprite As Int
Define YCoordinateofMainSprite As Int
```

You may want to check with a 6th former that you have done this bit properly, as if it is written in the wrong place, your program will not work.

- Now we can use the variables x and y to move around our sprite. If the right arrow is pressed, we want our sprite to move to the right, which means we want its x co-ordinate to increase. Therefore, we can say that when the right arrow is pressed, x should increase in value by 1. This can be written, in KPL as

```
XCoordinateofMainSprite = XCoordinateofMainSprite + 1
```

This in itself will NOT move the sprite, it simply changes the number stored in variable 'x'. Therefore, we have the sprite, so its x co-ordinate is the same as the value in our x variable. This is done by saying

```
MoveSpriteToPoint("MainSprite", x, y)
```

Enter this code in the `CheckForAnArrowKey` method, but put it AFTER all of the IF loops.

8. If you run your program now and press a key, you will probably find that it does something a bit weird...The sprite moves when you push the first key, to the top left corner, but then doesn't move when you push any other keys. To stop it from jumping to the top left corner, add the following code to the beginning of Method `Main()`.

```
XCoordinateofMainSprite = 20  
YCoordinateofMainSprite = 200
```

The reason the sprite does not continue moving is because our procedure to check for key presses is only run once. We need to keep it running, again and again, so we keep asking "has an arrow key been pressed". For this we need another global variable. Using what you did in step 6, create another global variable, called `HasCrashed` of type `Bool`. Type `Bool` means the variable is either true or false. We will use this variable to keep the computer asking if an arrow key is pressed, as long as there has not been a crash.

9. We now want to add a bit of text to our `Main` Method, which says that it should keep checking for an arrow being pressed, as long as the UFO has not crashed. For this we can use a while loop, which uses the following algorithm

```
While Variable = Result  
    //Code added here to say what it should do when variable = result  
End While
```

Here, your program will repeat the code which is between `While.....` and `End While`, when the `variable = result`. In our case, the variable is '`HasCrashed`' and the result is '`False`', as we want the loop to repeat while the user has not crashed. Try to see if you can use this information to write your own while loop in Method `Main()`

10. Now add some code to your `While` loop which will call the procedure `CheckForAnArrowKey`. It would be wise to check with a 6th former if you have done it properly at this stage.
11. In our while loop, we are getting it to repeat code while the plane hasn't crashed. But the computer does not know if the plane has crashed by default! Therefore, we should tell it. Write, just above your while loop, a line of code which will tell the computer that the variable `HasCrashed` is equal to `False`.

```
HasCrashed = False
```

12. Now run your program again. This time, you should find that the UFO does move, but too much! If you push an arrow, the UFO moves too quickly to see, let alone control! The reason for this is that the computer processes the code so quickly that the program has asked if an arrow is being pushed (and processed the relevant code) hundreds of times, when you just tap the arrow. What we need to do is to get the program to slow down. This can be done by using the '`delay`' command. In your while loop add, add the following code to make the program pause for a small delay each time you press an arrow key.

```
Delay(Add a number here)
```

Experiment with different numbers to find a good number, which allows you to be able to control the UFO.

13. We now want to see if we have crashed into anything. Using the same ideas as in step 1 and 2, make a procedure called '`DetectACrash`'. Then call it from inside your while loop.

14. In our program, we are eventually going to detect crashes with either the edge of the screen, or a 'baddy'. At the moment, we are only going to just detect crashes with the wall. Using the same idea as in step 1 and 2, create a procedure called DetectWallCrash. From the DetectACrash procedure, call the DetectWallCrash procedure. We can also write a note to ourselves, that we have got to come back to the DetectACrash procedure, in order to write code which will detect a crash with a baddy. Programmers do that as follows

```
//TODO: A brief description of what you are going to write
```

15. We will now add code to our DetectWallCrash procedures. We currently have a 600 x 440 box and we want to detect if the UFO hits this box – if there was to be a collision with the top of left, the x or y co-ordinate would be less than 0. If it were to collide with the right, the x coordinate would be greater than 600 and if it were to crash with the bottom, the y coordinate would be greater than 400. In order to do this, we could either use a lot of if clauses after each other, or use 'or' in the if clause, such as below

```
If (a < b) Or (c < d) Or (e > f) Or (g < h) Then  
  
End If
```

We will use the following code (it should be 1 line long)

```
If (YCoordinateofMainSprite < 0) Or (XCoordinateofMainSprite < 0) Or  
(XCoordinateofMainSprite > 600) Or (YCoordinateofMainSprite > 440) Then
```

16. Add the following code into your If loop, so that the game will stop when you crash

```
HasCrashed = True
```

17. We now are able to move our UFO around the screen and detect is crashing with the sides, so we can start to introduce the baddies. Firstly, we need to create a **type definition** for baddies, we won't go into too much detail on these. Type the following into your program, just above where you have declared the global variables XcoordinateOfMainSprite

```
Structure BaddyStructure  
    Name As String  
    x As Int  
    y As Int  
    Speed As Int  
End Structure
```

What this means is that when we create our baddies, each one of them will have a name, an x coordinate, a y coordinate and a speed.

18. We are going to have 9 baddies in the game, from what we know so far, the only way to do that would be to write the following

```
LoadSprite("Baddy1", "UFO.gif")  
LoadSprite("Baddy2", "UFO.gif")  
LoadSprite("Baddy3", "UFO.gif")  
etc
```

19. However this is not a nice way, so instead we are going to be bright sparks and use **arrays**. Firstly, type the following at the top of your program, just below 'Define YCoordinateofMainSprite As Int'

```
Define Baddy As BaddyStructure[9]
```

The square brackets at the end means that the computer is automatically going to create variables called Baddy1, Baddy2, Baddy3 for us, all the way up to 9....this is just one line of code, but yet it has created 9 variables! We declare them of type BaddyStructure, because we want each baddy to have its own speed, coordinates and name.

20. If you have got here, you have got over the worst part of making the game, if not, speak to a VI former as the past 2 stages are REALLY important.
21. Use the ideas of steps 1 and 2 to create a procedure called 'LoadBaddies' and then call it from Method Main(), just after DrawBackground()
22. The next step you may think it a bit daft, we're now going to tell the computer that Baddy1 is called Baddy1 and that Baddy2 is called Baddy2 etc. We are also going to set the initial speed of the baddies, make them visible and make them a bit smaller. Type the following code in your load baddies procedure.

```
Define Count As Int
For Count = 1 To 9
    Baddy[Count].Name = "Baddy" + Count
    LoadSprite(Baddy[Count].Name, "Plane.gif")
    ScaleSprite(Baddy[Count].Name, 0.7)
    ShowSprite(Baddy[Count].Name)
    Baddy[Count].Speed = (Random(1,3))
    FlipSpriteHorizontally(Baddy[Count].Name)
Next
```

23. Now run your program and you will hopefully see the planes, all in the top left corner of the screen. We will now animate them. Use the same ideas as in steps 1 and 2 to create a method called AnimateBaddies and call it from the while loop in Method Main().

24. In AnimateBaddies(), copy out the following.

```
Define Count As Int
For Count = 1 To 9

    Baddy[Count].x = (Baddy[Count].x + Baddy[Count].Speed)
    MoveSpriteTo(Baddy[Count].Name, 640 - ( Baddy[Count].x), Baddy[Count].y)
    If Baddy[Count].x > 750 Then
        Baddy[Count].x = 0 - Random(0,200)
        Baddy[Count].y = Random(0,480)
        MoveTo(0,480)
        Print(CrashCount)
    End If
Next
```

25. Now if you run the game again, you should have the baddies moving from right to left whilst you avoid them
26. To finish off, we can add the code to detect a crash with a baddy. To do this, create a procedure called DetectBaddyCrash() (using the ideas from in steps 1 and 2) and then call it from within the while loop in Main Method()
27. Add the following code to the procedure.

```
Method DetectBaddyCrash()  
  Define CrashedWith As String[]  
  CrashedWith = GetSpritesThatIntersectWith("MainSprite")  
  If ArrayLength(CrashedWith) > 0 Then  
    HasCrashed = True  
  End If  
  
End Method
```

28. Play the game a few times
29. Finished? Now try to look through steps 22 and 24 so that you understand what is happening to all the variables and how you get the effect that there are hundreds of baddies which fly at you.
30. Finished again? Now delete the procedures AnimateBaddies() and LoadBaddies() from your program and try to rewrite them without looking at this guide or anything else.