

## 2DIY ActionScript FAQ

For the latest version of this document please go to

[http://support.2simpleweb.com/public/docs/FAQ/2DIY\\_ActionScript.pdf](http://support.2simpleweb.com/public/docs/FAQ/2DIY_ActionScript.pdf)

**Q: Where can I find examples of using Actionscript code with 2DIY?**

**Q: Can I make the snake have a shorter tail?**

**Q: Can I add objects to the snake's tail when it eats them?**

**Q: Where can I find examples of using Actionscript code with 2DIY?**

- View the latest 2DIY user guide (which contains some code examples) here: [www.2simple.com](http://www.2simple.com) > support > downloads & updates. Also:

- <http://www.2simple.com/2diy/examples/>
- [www.2diyarchive.co.uk](http://www.2diyarchive.co.uk), a site (independent from 2Simple) created by Simon Widdowson from Porchester Junior School in Nottingham.
- Download examples from here: [http://support.2simpleweb.com/public/2DIY/as\\_examples.zip](http://support.2simpleweb.com/public/2DIY/as_examples.zip)

Advanced option - you can get more insight into how 2DIY uses ActionScript as follows:

- Go to C:\Program Files\2Simple Software\2DIY and edit the "2diy.ini" file using Notepad.
- Set "outputCodeAsFile=True" (near the bottom of the list). Save and close the ini file.
- Open 2DIY and start an activity or game and press the play button.
- Go to your My Documents folder and you will now find a file named "2diy code.txt". This file contains some of the ActionScript code which 2DIY uses when creating your swf file. You will find some of the variables and functions 2DIY defines, and you can use these in your own code blocks as well.

**Q: Can I make the snake have a shorter tail?**

A: Yes. In the Snake game, drag a sun object to the main area on screen. Edit the sun's animation and choose "Adv". Add the following ActionScript code.

```
for(n=6;n<=20;n++) {  
    ob=eval('_root.p'+n);  
    ob._x=-100;  
    ob._y=-100;  
};
```

This will hide all except the first 5 segments of the snake's tail. The first line provides a loop which cycles through from n=6 to n=20. The 2<sup>nd</sup> line sets which segment of the tail is being changed. There are 19 tail segments in total, named p2, p3...p20. As with all other variables, they need to be accessed with "\_root." in front. p2 is closest to the snake's head. The 3<sup>rd</sup> and 4<sup>th</sup> lines set the x and y position of the tail segments to negative numbers, ie off the screen. The above code could have been placed in the start-up section, for code to run once only at the start of the game. This would have worked for the first life of the snake, but when the snake dies, the code would not get executed again and the snake would have a full length tail again. This is why I placed the code in the sun animation; it is not ideal because its inefficient to set the position at every moment, but because it is only for a small number of objects, the effect is not noticeable. An alternative to setting the tail segment x and y positions to be off screen could be to set the segment to be invisible or have width zero. This does effectively hide the segment, but the snake will still die when it collides with the invisible segment.

**Q: Can I add objects to the snake's tail when it eats them?**

A: Below are some approaches you could take. The code samples described below can be downloaded from <http://support.2simpleweb.com/public/2DIY/snake/>. Please note that the code below is quite complex and definitely not for beginners.

1. The first approach is to leave the tail as is, and cause any eaten objects to copy the original tail segment's movements.

- a. Add to the start-up code block:

```
var numCaught:int = 0;
var caughtMonsters:Array=new Array();
```

The first is a variable which stores the number of objects the snake has caught. The 2<sup>nd</sup> is an array of objects which have been caught.

- b. Add to the collision animation of each monster object you add to the canvas:

```
var alreadyCaught = false;
for (n=0;n<_root.numCaught;n++) {
    if (_root.caughtMonsters[n] == this) {alreadyCaught = true;}
}
if (!alreadyCaught) {
    _root.caughtMonsters[_root.numCaught] = this;
    _root.numCaught++;
}
```

We use monsters as objects for the snake to collect because monsters have a collision animation code block but suns and apples do not. When the snake collides with a monster, 2DIY loops through the caughtMonsters array and checks if the monster has already been caught previously. If it has been caught previously, then it is ignored. If it has not been caught previously, it gets added to the caughtMonsters array.

- c. Add to a sun animation:

```
for (n=0;n<_root.numCaught;n++){
    segToFollow = n+2;
    ob = eval('_root.p' add segToFollow);
    _root.caughtMonsters[n]._x = ob._x;
    _root.caughtMonsters[n]._y = ob._y;
}
```

This loops through all the caught objects, and makes them copy the movement of the existing tail segments.

2. The second approach ignores the tail segments completely and causes the caught objects to follow the snake's head using their own follow algorithm.

- a. We start by hiding ALL the original tail segments of the snake, using the code mentioned above, but this time with n starting at 2 (add this to a sun animation.)

```
for(n=2;n<=20;n++) {
    ob=eval('_root.p' add n);
    ob._x=-100;
    ob._y=-100;
};
```

- b. Add the same start-up code and monster code as in the first approach above, but change the sun animation code for what the caught objects should do about following the snake. Use the code below:

```
for (n=0;n<_root.numCaught;n++){
    _root.caughtMonsters[n]._x += ((_root.player._x - Math.cos(_root.player._rotation
* Math.Pi/180)*25*(n+1)) - _root.caughtMonsters[n]._x) / (8+n*8);
    _root.caughtMonsters[n]._y += ((_root.player._y - Math.sin(_root.player._rotation
* Math.Pi/180)*25*(n+1)) - _root.caughtMonsters[n]._y) / (8+n*8);
}
```

This code works out the x and y position of each caught monster. It starts off from the position of the snake head ("\_root.player"). It then takes the cos or sin of the angle of rotation of the snake, and multiplies this by the order in which the object was caught. The

effect this has is to produce a line of objects behind the snake's head, with the line having the same angle as the snake's head itself. This is the target position for the caught object, but it is not moved to that position immediately; rather, it is moved there gradually, by taking a difference between the target position and the current position, and dividing that by a number. The number that is used for the division is also dependent on the order in which the object was caught; the further away an object is from the snake's head, the longer it will take to get to its target position, giving a "snake-like" movement.

3. The third approach is almost exactly the same as the second approach, except we can use the same idea in the journey screen and not have to worry about the original tail at all. The movement of the car is also more fluid and suited to the new tail algorithm we have created. Use the same startup code and monster collision code as above, but change the sun animation to the following (just replaces "player" with "car"):

```
for (n=0;n<_root.numCaught;n++){
    _root.caughtMonsters[n]._x += ((_root.car._x - Math.cos(_root.car._rotation *
Math.PI/180)*25*(n+1)) - _root.caughtMonsters[n]._x) / (8+n*8);
    _root.caughtMonsters[n]._y += ((_root.car._y - Math.sin(_root.car._rotation *
Math.PI/180)*25*(n+1)) - _root.caughtMonsters[n]._y) / (8+n*8);
}
```

2Simple Software  
[support@2simple.com](mailto:support@2simple.com)  
020 8203 1781  
Last updated 21 Apr 2010