

# **Simulated Artificial Intelligence**

by: Lucas Junqueira

Files needed for this article

- [Pick](#)
- [Pattern](#)
- [ichase](#)
- [Chase](#)

The objective of this tutorial is to show how to develop artificial intelligence algorithms to be used in IMSI Multimedia Fusion applications. However, it does not intend to cover everything about AI, just give ideas that can help the developers to make their own algorithms.

Keep in mind that the AI algorithms used in a game are very specific to the situations found at it, so it's better to see this tutorial as a guide to help you develop your own code. Also, you can not be afraid of mathematical functions, as long as they can help incredibly, but you also do not need to study math just for it as the standard game situations can be described by very simple mathematical expressions (one of the examples shown here uses a function to illustrate this).

In order to show how artificial intelligence can be implemented in Multimedia Fusion, some algorithms will be developed here:

1. the chase/escape algorithm,
2. the improved chase/escape algorithm,
3. the pick algorithm,
4. the pattern algorithm.

Also, you can find some tips about creating enemy personalities.

This tutorial is based on the texts below, and their reading is highly recommended:

- Teach Yourself Game Programming, André LaMothe, Sams Publishing

- Flights of Fantasy, Christopher Lampton, Waite Group Press
- IMSI Multimedia Fusion Reference Manual, IMSI



The chase/escape algorithm



Remember that games with villains doing everything they can just to get you? Well, this algorithm show how to make MMF control active objects to act like that. You can open the "chase" file to see this.

First it was needed to choose the graphics to be used in the example. Pac-man and his old friend, the ghost, seemed ok for this job...

Start by making the player object. In this example, pac-man was given the eight directions movement. Than create the enemy object. In this case, it was given the bouncing ball movement, necessary to the algorithm adopted, as you will see soon.

Now, take a look at the behavior#1 of the enemy object in the example file you downloaded. As you can see, it is oriented to change its movement every time, looking in the direction of the player. That is the chase personality of the ghost. By hitting the space bar, you change the way the ghost acts. As you can see, now it is oriented to change it's direction every time in order to run away from the player. That's the escape personality.

That's it! A very simple algorithm that create a very simple movement, of course.



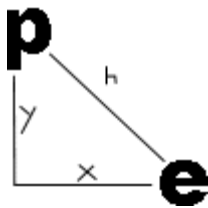
The improved chase/escape algorithm



If the enemy looks for the player so hard, he'll never stand a chance! The improved version of the chase/escape algorithm considers the line of sight of the ghost. He'll chase/escape the player only if he can "see" it. Open the "ichase" file to see this.

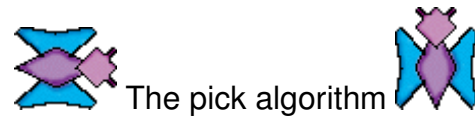
To implement this, a simple mathematical formula was used. It calculates the distance between the player and the ghost. If this distance is less than 150 pixels (this was defined as the "sight limit" of the enemy), the ghost acts considering the chase/escape algorithm. If not, it continues its random bouncing ball movement. The big transparent circle was put in the screen just to illustrate the sight area of the enemy, it does nothing besides this.

The mathematical formula used in this case is the Pitagoras' Theorem for triangles:



$h = \text{distance between player and enemy} = \text{SQRT} (x^2 + y^2)$

Now give a look at the behavior#1 of the ghost in the example you downloaded.

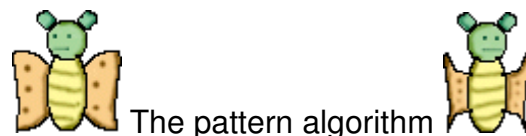


Have you ever noticed that when you use a path-oriented object in MMF, and put some of them at the same time in the screen, they act exactly the same way? Of course you noticed, in fact, that's the idea of this kind of computer controlled movement. But when you use this, you give the player the idea of a deterministic game universe, sometimes reducing a lot the exciting. This algorithm shows how to keep using the path movement, but changing the way the objects move in a very easy way.

Open the "pick" file to see this. It shows some insects flying on the screen. At the left side, you can see the deterministic movement, and at the right, what can be called the random one.

To do this, the "pick object" action was used in the events of the frame. In regular time intervals, some insect objects are picked at random, and its speed is changed (not much, just the sufficient to give the idea we want). You can see this at the events screen of the example file you downloaded.

Well, you can, of course, change any other property of the objects (like animation and so on...) in order to give the player this random feeling.



Despite the simple look of the example file, this algorithm is the most powerful one developed in this tutorial.

Open the "pattern" file to see this. You can see two insects (that was called mickeyflies - really don't know why) that move in a very strange way, sometimes the same way, sometimes not.

The idea of this algorithm is to use the arrays to give instructions to the application. In this case two arrays were used. You can see their contents at the table below:

position	array: horizontal movement	array: vertical movement
1	1	2
2	1	2
3	1	2
4	1	2
5	1	2
6	3	4
7	3	4
8	3	4
9	-1	-1

At the behavior#1 of each array, a file containing the information of this table is loaded at the start of the frame. Now take some time to see the behavior#1 of the mickeyflies (any of them, as long as they are just clones). To create this information files, a simple array editor was used. It is included in the file downloaded (along with the two arrays used), so you can modify it to fit your needs.

You can see that, every second, MMF is instructed to read a value of the current position of the array in use. Than, the position an the animation direction of the object is updated according to the number read in the array. The next table shows the relation between the number read and the MMF action.

number read	effect in position	effect in animation direction
1	position x = old x + 10 pixels (move right)	change to right
2	position y = old y + 10 pixels (move down)	change to down
3	position x = old x - 10 pixels (move left)	change to left
4	position y = old y - 10 pixels (move up)	change to up

When MMF finds a -1, it recognizes it as the end of the array, so it resets the array position and randomly chooses another one to use.

To improve the usage of this algorithm, you can use some of the following ideas:

1. by reducing the number of pixels added in the position, you can achieve smooth movements,
2. increasing the number of array positions, adding y columns and so on will also be needed in order to create complex patterns (witch is the main goal of this algorithm),
3. you can also use the patterns to create dynamically alterable path movements that can be changed according to your game situation or player activity,
4. the patterns can also be used to control features other than movement, like

animation or sound reproducing.



Tips about personalities



You can use any of the algorithms above to simulate enemy personalities. Having enemies that behave different from each other is a great feature of a game. You can make some ghosts with different line of sight in the chase/escape algorithm, or flies with speed increase/decrease of their own.

But, as said before, the pattern algorithm is the most powerful of them. When you choose the next pattern, you don't need to do it randomly. Take some time to create your patterns, so you can make your objects act in a very complex way, according to the individual personalities or player interference. You don't need to make the -1 the only way to get out from an array, some dangerous situations could force your character to change its behavior (change the pattern). Of course, lots of patterns are needed in order to create a complex game, but that's the way it is...

Copyright © 2000 Lucas Junqueira. All rights reserved.  
Please **do not copy or re-use** without written permission.