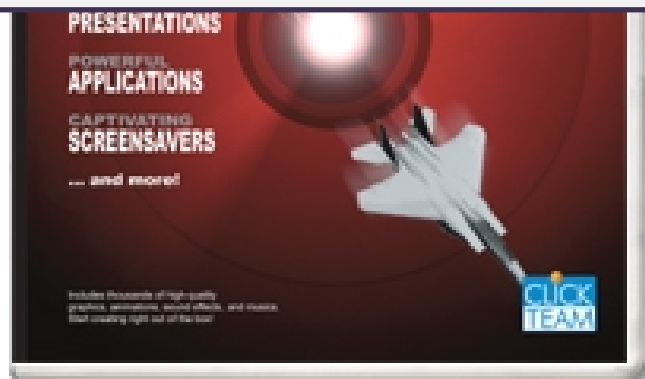


31. 10. 2008



PART 1

THE CLICKER'S GUIDE TO FASTLOOPS



The Clicker's Guide to Fastloops | Written by Popcorn

Contents

Who should read this tutorial	2
Introduction to fastloops.....	2
What are fastloops?	3
How to create a fastloop	3
Important note about fastloops.....	5
Basic examples	5
Counting	5
Create many objects.....	6
Moving objects	6
Bullet with perfect collision detection	7
ID's and looping through objects	9
Spreading values and looping through each instance of an object	9
Find the object with the highest alterable value	9
Detect end of platform for multiple enemies	11
Nested loops.....	12
Fill a grid with objects.....	13
Placing objects in a movable circle.....	15
End of part 1	17
About the author	17

Who should read this tutorial

This tutorial is for the intermediate MMF2 and TGF2 users who want to expand upon their game creating. It might also be a source for you if even if you already know how to use fastloops since the examples in this tutorial vary from beginner to advanced. You should already be quite comfortable with the program before starting learning about fastloops.

Introduction to fastloops

Fastlooping is essential in game and application making. Every person that starts to learn a programming language or a script based tool knows what a loop is and how to use it, simply because it isn't easy to make anything without it.

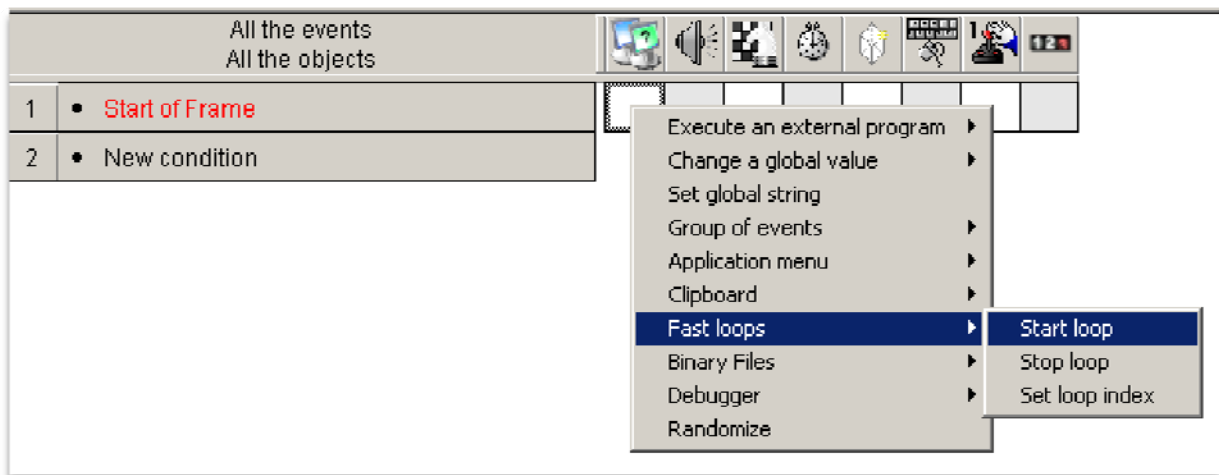
"Unfortunately", MMF-users are spoiled with in-built functionality. MMF has already taken care of many things that would have required fastlooping in a programming language. Therefore, new MMF-users often don't see the benefits of fastloops before they actually try it and understand how it works.

What are fastloops?

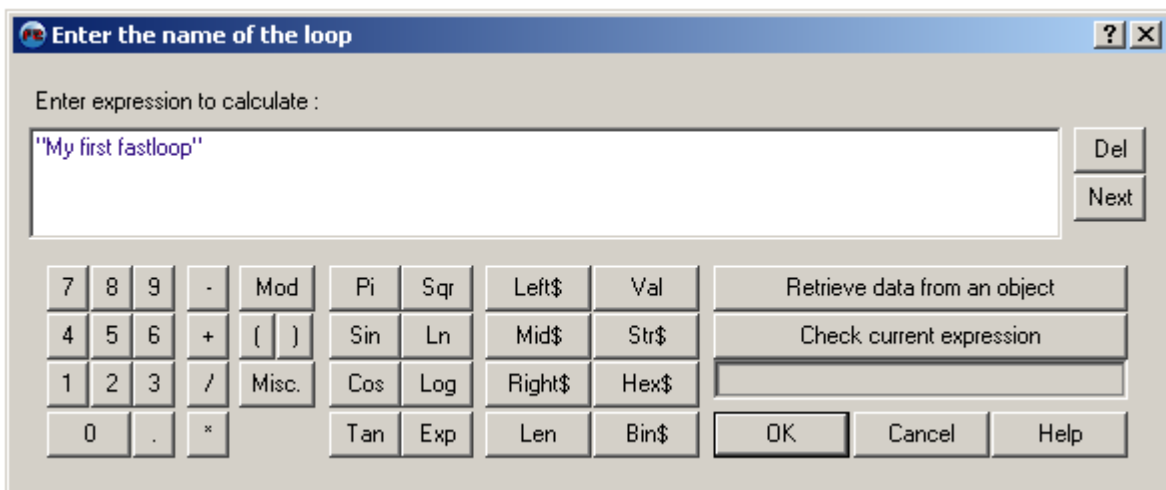
Simply put, a fastloop is a set of events that runs as many times as you want before anything is drawn on the screen. I will not write pages up and down on what a fastloop is and how fastloops work in general. That topic is already taken care of in other tutorials by other people. I will take another approach and jump straight into making examples. The first ones will be very basic so you don't actually need to know anything about fastloops before you start. Each example is explained, so hopefully you'll learn as you progress through the examples.

How to create a fastloop

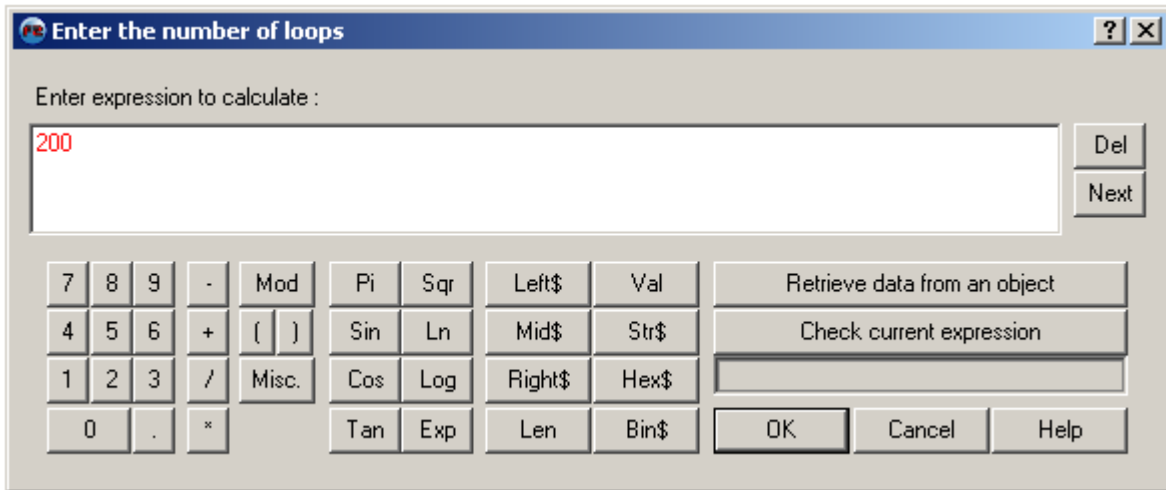
Enter the event editor. To start a new fastloop, first create the condition(s) that will execute the loop, for instance **Start of Frame**. The action to start a fastloop is found under the Special Condition icon.



Give the loop a name and press OK.

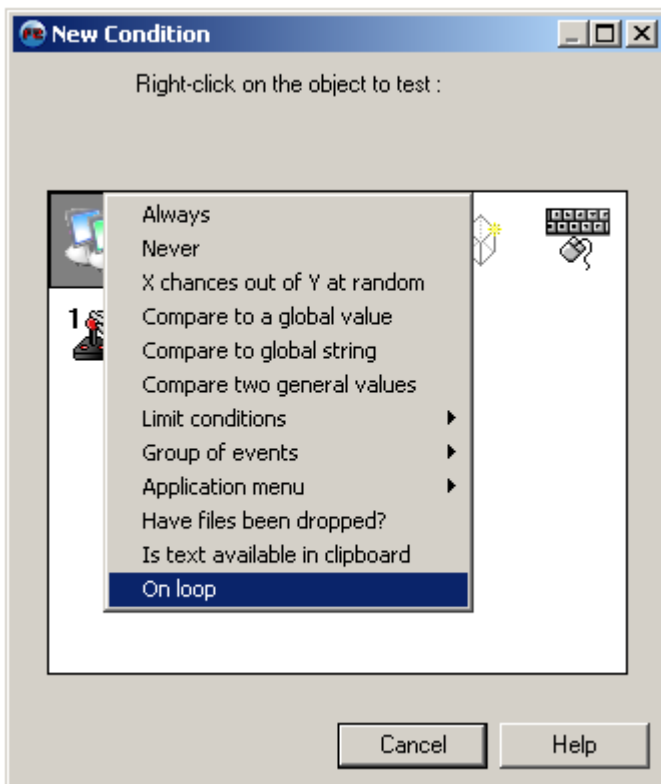


Then specify how many times the loop shall run.

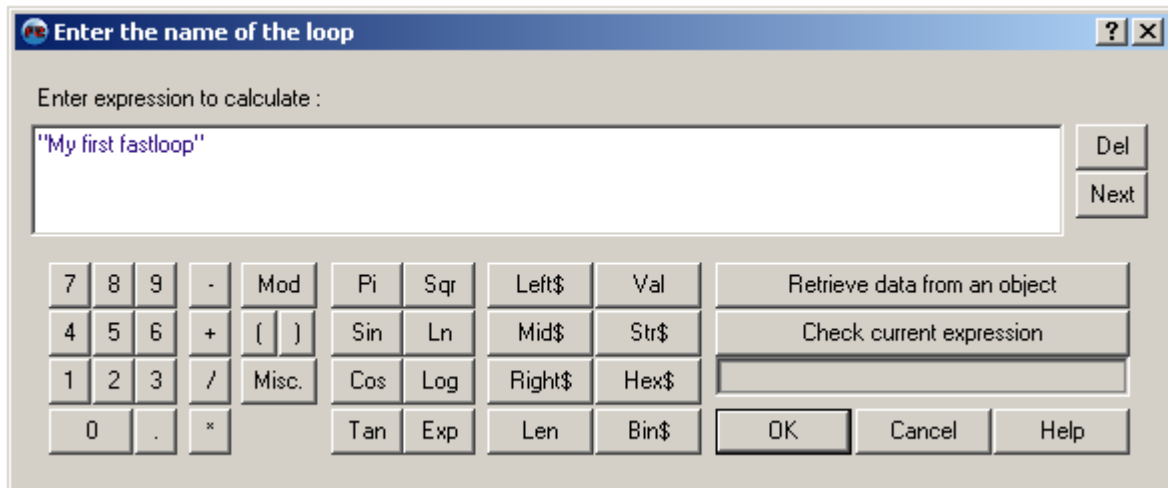


This will make the loop “My first fastloop” run 200 times.

To do an action in a loop, create a new event and add the **On loop** condition.



Enter the name of the fastloop you created.



Then add the action that should be executed in the loop.



When you start a loop, MMF will scan through the event editor for all **On loop** conditions for that loop. The events which have the **On loop "My first fastloop"** condition will be run as many times as the loop is running, in this case 200 times.

Important note about fastloops

Anything that happen inside a loop will not be available from outside the loop. If you always run a loop 10 times and on each loop you add 1 to a counter, the general loop-cycle will count 10, 20, 30 etc.. The counter will never read 13, 17, 24, 31 etc... If you want to do an action when the counter is 27, you will need to do an **On Loop** condition. Also, once a loop is started, all **On Loop** conditions for the loop will be executed before any other actions in the event and before any other event in the normal loop-cycle.



Basic examples

Counting

In this first example we will make a fastloop that will run ten times when we press a button. Each time the loop executes it will add 1 to a counter. Try making the events below and see what happens!

You need:

A counter set to display numbers or text

- **Upon pressing any key**
 : Start loop "count" 10 times
- **On loop "count"**
 : Add 1 to Counter
- New Condition

Run the frame. Do you see that the counter increase by 10 every time you press a button. Why does it do that? Because we told it to count to 10 before it draws the screen. It adds 1 ten times, and then it draws the screen.






Fastloops simply let you repeat the event many times before the screen is redrawn. Not useful you say? Extremely useful, I say. Just wait...

Create many objects

Sometimes we want to create lots of objects at once. This little example will show you how to make a cute little fireworks by creating lots of objects when you click the mouse.

You need:

A small active object with movement set to Pinball and with Create at start deactivated.

- **User clicks with left button**
 : Start loop "fireworks" 50 times
- **On loop "fireworks"**
 -  : Create  at (320,240) layer 1
 -  : Set X position to XMouse
 -  : Set Y position to YMouse











Simple, but neat, eh? It creates 50 active objects. At default each is given a random direction.

Moving objects

Custom movements would be very hard if not impossible to make without loops. In real programming built-in movements are non-existent, so every programmer learns how to make their movements with fastlooping.

You need:

An active object

- Repeat while "Right Arrow" is pressed
 : Start loop "move right" 5 times
- Repeat while "Left Arrow" is pressed
 : Start loop "move left" 5 times
- Repeat while "Up Arrow" is pressed
 : Start loop "move up" 5 times
- Repeat while "Down Arrow" is pressed
 : Start loop "move down" 5 times
- On loop "move right"
 : Set X position to X()+1
- On loop "move left"
 : Set X position to X()-1
- On loop "move up"
 : Set Y position to Y()-1
- On loop "move down"
 : Set Y position to Y()+1

This will move your object in a given direction. How many times it loops decides how fast the object will move. In this example the object will move 5 pixels when you press an arrow key. It will continue to do that until you release the key.

This might not seem too impressive at first. In the example [Bullet with perfect collision detection](#) we will add collision detection to the object which tests for collision inside the loop. Then you'll see how useful this method really is.

Bullet with perfect collision detection

With perfect collision detection your bullet will hit the target no matter how small the bullet or the target object is. If both objects are about 3 pixels wide and long each, and the bullet is moving with let's say 10 pixels each frameloop, normally there is a big chance the bullet will skip the target by "jumping" over it since it is only testing every tenth pixel.

Here I will show you how to do perfect collision detection for a bullet in a 2D spashooter. We will move the bullet like we move the object in the previous example. I'll do a very basic example for a spashooter where the player can only shoot upwards. The difference is that we add collisiondetection inside the loop. That means that we can test for collision on every pixel even if it seems like the bullet are moving with 10 pixels each frameloop, since the screen only refreshes that often.

So basically what we do is this:

- Move bullet one pixel
- Test for collision

- Move bullet one pixel
- Test for collision
- Etc...
- Draw the frame

Instead of the normal method without perfect collision detection:



- Move object 10 pixels
- Draw the frame
- Test for collision

So let's make this!

You need:

3 active objects:

- **Player ship – 8 direction movement or a custom one**
- **Enemy – can be static**
- **Bullet – static, with Create at start deactivated**

- Upon pressing "Space bar"
 -  : Create Bullet at (0,0) from Player ship (action point)
- Always
 -  : Start loop "Move bullet" 10 times
- **On loop "Move bullet"**
 - + Bullet is overlapping Enemy
 - Bullet : Destroy
 - Enemy : Destroy
- **On loop "Move bullet"**
 - Bullet : Set Y position to Y("Bullet")-1

If you want to move the bullet faster, increase the number of loops, and if you want it to go slower, decrease the number of loops. It is important that the object only moves one pixel inside the loop to retain the perfect collision detection.

ID's and looping through objects

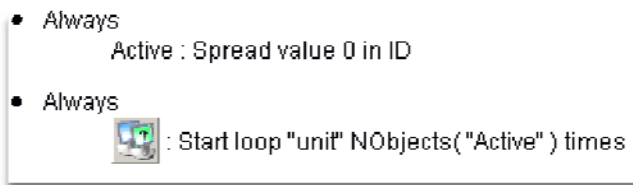
Spreading values and looping through each instance of an object

This should be a fundamental thing in every game. We will basically give each instance of an object a unique id so that we can test for each object of the same type in a loop. This has extremely many uses.

You have probably seen the Spread value action. This action simply numbers the objects. If you have 4 active objects called ENEMY and spread value 0 to ENEMY, the first ENEMY gets value 0, the next 1, the next 2 and the last one 3. This is a very useful feature when we want to loop through all the ENEMY objects.

What we do is that we start a loop and run it as many times as there are ENEMY objects. In this case the loop would run 4 times. On each loop we test the ID of the object. The ID is just an alterable value that we have spread value 0 into. If the ID of the object equals the loopindex, it means that this object is tested for now. The first time the loop runs, the loopindex is 0. MMF tests every ENEMY if the value ID is 0. This will only be true for one object, and that object is the one that will be tested for this time. On the next loop the loopindex is 1, and now it is the ENEMY object with ID set to 1 that is tested for.

A common thing for all ID loops is that we first rename one of the alterable values to ID. Then we have two events. In one event we spread the values, in another event we start the loop. You can never start a loop in the same event as you spread values, that simply doesn't work.



If you are sure that none of the objects will be destroyed or there will be more objects of that type, then you can replace Always with Start of Frame. I like to use Always because then it makes sure that the objects always have a ID from 0 to (number of objects).

The following examples will show you a few basic uses for ID looping.

Find the object with the highest alterable value

This example will show you how to loop through each instance of an object and remember which one had the highest alterable value.


You need:

1 active object, place several duplicates in the playarea:

- Rename alterable value A to ID
- Rename alterable value B to MyValue

2 counters

- HighestFound
- GetFixed

- **Start of Frame**
Active : Set MyValue to Random(2000)
- Always
Active : Spread value 0 in ID
- Always
 : Start loop "unit" NObjects("Active") times
- **On loop "unit"**
 - ID of Active = LoopIndex("unit")
 - MyValue of Active > value("HighestFound")
HighestFound : Set Counter to MyValue("Active")
GetFixed : Set Counter to Fixed("Active")
- Fixed value of Active = value("GetFixed")
Active : Set semi-transparency to 50

First we give each object a random value and a unique ID.

Then we start the loop named "unit" and loop it as many times as there are objects named Active.

MMF loops through all Active and tests if the alterable value MyValue is higher than the counter HighestFound. If it is, the counter gets the object's value and the counter GetFixed gets the fixed value of the object. The fixed value is a random unique number for each object in the game.

The first time the loop is running, HighestValue will be 0, so the first object tested for will be the one with the highest value till now, if its value is higher than 0.

Let's say its value was 900. The next object in the loop has a value of 500. MMF test if MyValue is higher than HighestFound, but this time it isn't, so the counter remains the value of 900.

The third object in the loop has the value of 1500. This time the counter HighestFound sets to 1500 and the counter GetFixed gets the fixed value of this object.

When the loop is over, all objects have been counted for and we know which one has the highest value. We can find this object by comparing its fixed value with the counter GetFixed.

Make sure that since we are not defining which Active we are referring to, you can not use the *Compare two general values* condition in this event. You have to use the Active object's own

Compare to fixed value condition instead. As you see, this comparison is outside the loop and doesn't happen before the loop is over. Then we can do an action on this one object!

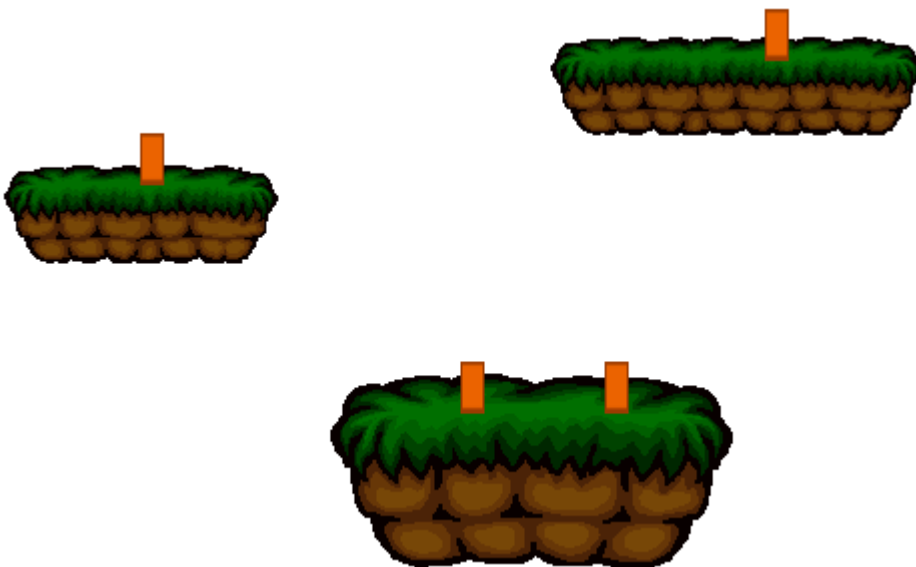
Do you start to see how useful fastloops are? This would be impossible to do without them. Of course there are extensions that do this job, but don't you think they use loops as well? ☺

PS!

If you want to find the object with the lowest value, add another counter and give it a way too high initial value. Then make a new event similar to the one testing for the highest value, but this time update the new counter if an object's value is lower than the counter's value.

Detect end of platform for multiple enemies

The title may be a bit vague, but I'll try to explain what I mean. You have a level with several platforms. On each platform there are enemies walking. When an enemy comes to the edge of the platform he turns around. We are using the collision mask conditions to detect the edges. This is easy when there is only one enemy. The trick is when there are multiple enemies, each one checking if they are near the edge of the platform. The method I'll show does not use any detectors.




The orange boxes are the enemies. Make sure to set their hotspot on the bottom, or the code wouldn't work unless you edit it for your needs.

You need:

1 active object, place several duplicates in the playarea.

- **Name it Enemy**

Some backdrops set to obstacle. Place the enemy objects on these obstacles as shown in the picture.

- Enemy: internal flag 0 is on
Enemy : Set X position to X("Enemy")+1
- Enemy: internal flag 0 is off
Enemy : Set X position to X("Enemy")-1
- Always
Enemy : Spread value 0 in ID
- Always
 : Start loop "detect edge" NObjects("Enemy") times
- **On loop "detect edge"**
 - ID of Enemy = LoopIndex("detect edge")
 - Enemy: internal flag 0 is off
 - **X** X Left("Enemy")-7,Y Bottom("Enemy")+3 is an obstacle
Enemy : Set internal flag 0 on
- **On loop "detect edge"**
 - ID of Enemy = LoopIndex("detect edge")
 - Enemy: internal flag 0 is on
 - **X** X Right("Enemy")+7,Y Bottom("Enemy")+3 is an obstacle
Enemy : Set internal flag 0 off

You see that we loop through each Enemy and tests if it is near an edge. There are one event for left and one for right. This is a general comparison and has to be done in a loop like this in order to work for multiple objects.

Nested loops

A nested loop may sound complicated, but really it isn't. It simply runs a loop inside another one. This is very useful in many cases. The next example will demonstrate nested loops in a very simple way.

Fill a grid with objects

We are going to fill the screen with squares that are 50*50 pixels in size. Our screen is 640 * 480 pixels. Simple math tells us that we need 13 rows and 10 columns with square objects. The objects shall be created instantly when the frame starts. The easiest way to do this is to run two loops. One loop that “move horizontally” and one loop that “move vertically”.

LoopX is started 13 times. On each LoopX loop we start LoopY 10 times. LoopY will create objects vertically. The created objects will get placed according to LoopX and LoopY's indexes. Since the object is 50*50 pixels in size, we set its x position to $\text{loopindex}["\text{LoopX}"] * 50$ and its y position to $\text{loopindex}["\text{LoopY}] * 50$.

The first time LoopY is ran, the loopindex of LoopX is 0, and the loop index of LoopY is 0. The calculation for the xposition is $0 * 50 = 0$, and the yposition is also $0 * 50 = 0$. The object is placed on (0,0).




The next time LoopY is ran, LoopX is still at 0, but LoopY is 1. The object will therefore be placed on (0,50). The following objects is placed on (0,100), (0,150) etc..

When LoopY is finished, the first column is filled with objects. But LoopX is not done yet. Its loopindex increases to 1. It starts LoopY 10 more times. Now the objects will be created at (50,0), (50,50), (50,100) etc..

When LoopX is finished, LoopY has been ran in total $13 * 10 = 130$ times. The screen is filled with 130 squares all placed nicely in a grid.

You need:

An active object, 50 * 50 pixels in size and with Create at start deactivated.

- **Start of Frame**
 : Start loop "LoopX" 13 times
- **On loop "LoopX"**
 : Start loop "LoopY" 10 times
- **On loop "LoopY"**
 : Create Active at (0,0) layer 1
Active : Set X position to $\text{LoopIndex}("LoopX") * 50$
Active : Set Y position to $\text{LoopIndex}("LoopY") * 50$

That was not difficult at all, was it? :D

PS!

It's a good rule to never hardcode any values if it can be avoided. So instead of starting LoopX 13 times and LoopY 10 times, start LoopX `ceil((Frame Width/50.0))` times and LoopY `ceil((Frame Height/50.0))` times. Ceil will round the value upwards, so that we are sure that we fill the entire screen. Remember to add a decimal in the calculation like I did, this way the result will be a float value and not an integer which is the default. Read more about float, int and rounding in other articles or tutorials ;)

Ok, so we hardcoded the size of the square. You can even retrieve its width and height using the following expressions:

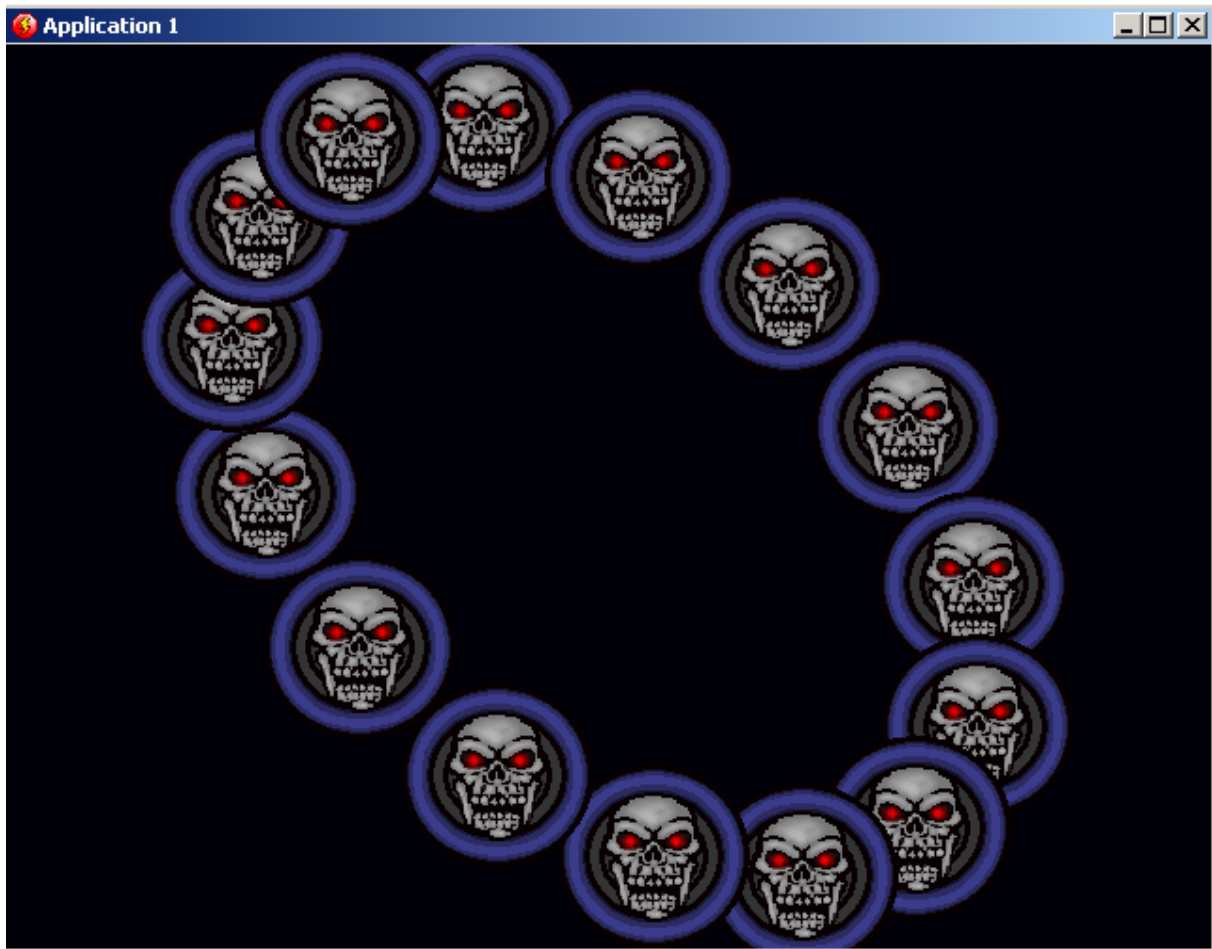
`Width = X Right("Active")-X Left("Active")`

`Height = Y Bottom("Active")-Y Top("Active")`

Placing objects in a movable circle

You need:

An active object with hotspot set at center and Create at start deactivated.



In the application shown above the user can control the circle by moving the mouse. To position an object in a circle we use the following two formulas:

$$X_{pos} = \text{Center} + \text{radius} + \sin(x \cdot 360.0/n)$$

$$Y_{pos} = \text{Center} + \text{radius} + \cos(x \cdot 360.0/n)$$



X is the id of the object and n is the total number of objects in the circle.

In MMF2 the formulas will look like this:

```
(Frame Width/2)+200*Sin(ID( "Active 21" )*360.0/NOjects( "Active 21" ))  
(Frame Height/2)+200*Cos(ID( "Active 21" )*360.0/NOjects( "Active 21" ))
```

Active 21 Is the name of the skeleton object I am using, and ID is its alterable value A.


As you probably have figured out already, we can put this directly into a loop. But first we need to make the loop that creates the objects.

- Upon pressing "Space bar"
Active 21 : Destroy
- Upon pressing "Space bar"
 : Start loop "Create circles" 3+Random(20) times
- **On loop "Create circles"**
 : Create Active 21 at (307,283) layer 1

The first event destroys any Active 21 already present. This is good when we want to create a new circle. The second event tells how many objects there will be in the circle. It creates at least 3 objects, and then a maximum of 19 objects more.

The third event creates the objects. It doesn't matter where you place the objects, they will be positioned later.

Here are the final 3 events:

- Always
Active 21 : Spread value 0 in ID
- Always
 : Start loop "Position objects" NOjects("Active 21") times
- **On loop "Position objects"**
 - ID of Active 21 = LoopIndex("Position objects")
Active 21 : Set X position to $(\text{Frame Width}/2)+200*\text{Sin}(\text{XMouse}+\text{ID}(\text{"Active 21"})*360.0/\text{NOjects}(\text{"Active 21"}))$
 - Active 21 : Set Y position to $(\text{Frame Height}/2)+200*\text{Cos}(\text{YMouse}+\text{ID}(\text{"Active 21"})*360.0/\text{NOjects}(\text{"Active 21"}))$

The first event as you know spreads an ID into the objects and the second event starts the loop. The final event positions the objects in the circle. The only value in the formula that is unique for each object is the ID. As you can see I put XMouse and YMouse in there in the formula. So now when you move the mouse, the circle will move, because the formula is always updated.

PS!

There is a great article about trigonometry in MMF2 / TGF2 in the article section of the Clickteam forums.

End of part 1

Do you now see how powerfull fastloops are? I hope that you have enjoyed this first article of which I plan to make a series, and I hope that you now are starting to be comfortable with basic fastlooping. Go through all the examples if you haven't already, and try experimenting on your own to make your own loops. Good luck!

About the author

This guide was written by Popcorn. My real name is Kjetil Nossun and I have been working with Clickteam's products since 1996. You will find me as a frequent user of Clickteam's forum at www.clickteam.com.

Please give me feedback about this tutorial and also tell all your friends about it!

E-mail: ilikepopcorn@hotmail.com