

You've got Spacemail!



You may not use this tutorial for any other purpose than learning, working or having fun... In other words: You can use this PDF tutorial for anything you'd like, as long as it doesn't involve both a hammer and a squirrel.

Koobare
marchewkowy@gmail.com

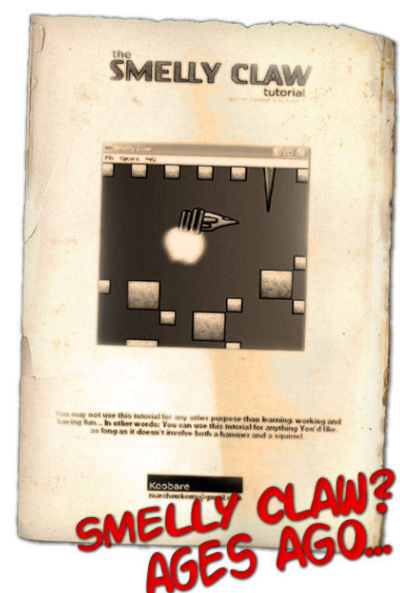
Welcome to yet another one of **Koobare's little tutorials**, teaching you – as always – how to effectively and efficiently use the best multimedia authoring tool ever – [Multimedia Fusion 2](#) by Clickteam! This tutorial is meant for beginner intermediates, people who already know a thing or two about MMF2, who have by now done their share of basic novice-level tutorials (you should read the “Interface Guide”, “Glob Wars”, “Smelly Claw” and “Risky Waters” tutorials before doing this one). Take a look at this simple lesson guide to help you decide which tutorials should be played with first (note that this is just a suggestion):

MMF2 Interface: Interface Guide + Image Editor Guide	<i>First time with MMF2</i>
Basics: Smelly Claw tutorial	<i>Beginners</i>
Game tutorials: Glob Wars and/or Risky Waters	<i>Beginners</i>
You are here → You've Got Spacemail!	<i>Beginner-intermediates</i>
Next: Castle Defender and/or Space Corsair	<i>Beginner-intermediates</i>
Next: Save & Load tutorial	<i>Intermediates</i>

In this tutorial we will create a nice-looking “Lander” type of game, with a speeding spacedroid trying to land on a roof of a tiny house, built on a distant and remote planet. We will use a few counters, active objects, object groups and event-grouping techniques to achieve a great-looking and good-playin’ game. During this tutorial we will create our own custom movement (based on the “bouncing ball” movement), which you may improve further on by yourself and use in your own “Lander”-inspired games.

Changes, changes, changes...

It's April of 2009... I've been writing tutorials for Clickteam for some time now. I've received tons of e-mails, tons were unfortunately lost by my malfunctioning incoming boxes (dunno' why, but it seems to still happen from time to time)... A lot – and I mean A LOT – of your suggestions have been implemented into my tuts, along with my own ideas and modernizing efforts... We've traveled a long way from my first tutorial, “Smelly Claw”, I guess we can all agree on that. Now it's time to introduce another one of your suggestions, that have sprung up in your e-mails from time to time – I'm going



to get my tutorials better organized as a structured course, as full-scale classes in MMF2 game design for beginners and intermediates. To do so, I'll divide all my upcoming tutorials into four series, named respectively **"Gamebuilders"**, **"Enhancing the Feel"**, **"Engine Works"** and **"Papa Koobare's Boot Camp"**.

"Gamebuilders" will be a series of tutorials in which the end-product will always be a playable game of *better-than-decent quality*, something that you can play after completing, or even use as a basis for your own project, as the core engine for your own game.



My second tutorial series, **"Enhancing the Feel"**, will be a bit different from "Gamebuilders". You won't find the same *"create a full game from scratch"*-approach in these tutorials, as they will focus on creating a few particular game elements that will be ready to easily implement into your own games and applications. The main idea behind this series is to help you enhance the gameplay experience of your players, to improve your game's overall "feel", by introducing additional gizmos, such as special graphical effects, night and day systems, attractive layouts for your menus and other extra elements that you can add to your own projects.

The third series, **"Engine Works"**, will be all about programming – how to create a basic enemy A.I. with as little scripting as possible, how to use the Pathfinding object, how to encrypt your savegames, how to create a simple save and load system using an array, that kind of stuff.

Finally, **"Papa Koobare's Boot Camp"** will be all about the basic basics, stuff that can be really interesting for first-timers, but would prove to be too *"been there, done that"* for anyone who has any experience with MMF2 whatsoever. This series will be designed specifically for all you nuggets who just took their first few steps into the fascinating world of Multimedia Fusion 2.

There will be even more changes coming in future tuts, but let's not be too hasty, we'll learn about them in proper time. As for now – let's learn something about the Spacemail Postal Service and why it's the best way to send letters and parcels to distant galaxies...

Spacemail – the postal service of the future!

We've all heard that before – the internet proved that the standard, old-fashioned postage service isn't needed anymore, right? Well... **Wrong!** In 2198 the project to build a spacecable internet link-up between Earth and its fourteen colonies failed, when an inch-wide meteor crushed into the 927,862,860,849,897 yard long wire, rendering it useless. Four years later the wireless connection went out as well – as it seems, sunspot activity had some influence on the broadcasted signal, turning all sent e-mails into ancient poetry and hamburger recipes (that – for some reason – utilized broccoli and watermelons as the two main ingredients).

Finally, after years of serving dreadful hamburgers, someone had the courage to stand up and ask: "so, why not do it in the traditional way?" In July of 2206, World President Bobby "Burger Savior" Melikeyapples signed the executive order to create a **new interspace agency**, tasked with delivering letters, packages, Christmas cards and tons of useless spam all across the galaxy. Since that day onward, the **Galaxy Postage Office's** mailbots rushed from one space system to another, enabling Earthlings to communicate with aliens and vice versa. No matter how many meteor fields they had to cross, no matter how many supernovas they had to evade – the mailbots were always on track, there was always something that had to be delivered, someone somewhere was always waiting for that tiny little card or envelope.



Our story begins twenty years later, on the bizarre **planet of Voopookoo**. It is here, under the ever-green sky, amongst the floating fuel-eating bubbles and strange giant balloons (chained to the ground, for some reason), that we find **Mailbot 7890-Kappa-91**, also known as **Raymond**, trying to deliver the last package in this rotation. He has limited fuel, his boost system is somewhat frenzied – really in need of a good tune-up – and he still needs to land on the top of that little pink cabin in the distance... The keyword here being **“to land”**, as opposed to (easier achievable) “crash” – blasting through the roof on top speed just ain’t gonna’ do it for him.

Your assignment – as a mailbot operator and a true friend of Raymond’s – is to help number 7890-Kappa-91 land safely and deliver his message. You’re going to use all your navigational skills to keep those engines going without using up the whole fuel tank, to make sure that Raymond is ready for the touchdown... And then to bring him down, once his speed is below “9” on the display... The tough part? The engineers that constructed this off-location remote control system have truly forgotten to put any numbers on your screen – you’ll have to trust your instincts and decide whether



the speed is low enough with the help of nothing but your hunch and a visual horizontal-bar display... What’s more, you better keep an eye on those odd bubbles floating in the air – it seems that they have a tendency to evaporate fuel from Raymond’s reserves! Well, **no one said that the life of a mailbot operator is easy**, right?

“You’ve got Spacemail!” – the basics...

Our game starts with a very short (few seconds in length, actually) **“intro”** of sorts – the mailbot appears closer to the “camera”, just to show off a few details and demonstrate to the player how he is built. During these few seconds the player has no control over the robot – it has to travel up the screen, disappear from the player’s view, and then return in a smaller form (it will actually be a different object, but – shush! – keep it for yourself), suggesting that it traveled a bit further away. Once it returns on-screen, the mailbot controls are unblocked and **the struggle between the player, gravitational pull and the fuel-burning begins**.

The main goal of the game is to land the mailbot directly on top of the roof – to do so, the player must keep the speed at *minimum* when approaching the landing zone and then press “space” to initialize the landing sequence. Our gamers control the droid’s movement with the help of the cursor keys, using up quite a lot of fuel with every press of the button – the longer they keep those buttons pushed, the higher the mailbot goes... But the fuel reserves become emptier and emptier as well. Furthermore, collisions with the floating “**killer bubbles**” take away lots of spacegas too – the bigger the bubble, the more fuel it “steals” from the mailbot’s reserves.

The game is lost when the fuel reserves go down to zero (the mailbot will fall down to the planet’s surface), when the mailbot touches any of the chained static balloons (not to be mistaken with the floating bubbles), or when he collides into the ground or one of the weird bonespikes that grow right out of it – there are quite a lot of ways to loose here, as you can see.

So... You now have the basic knowledge about the game we’re going to create... Time to turn the theory into practice... **Let’s get our game going, shall we?**



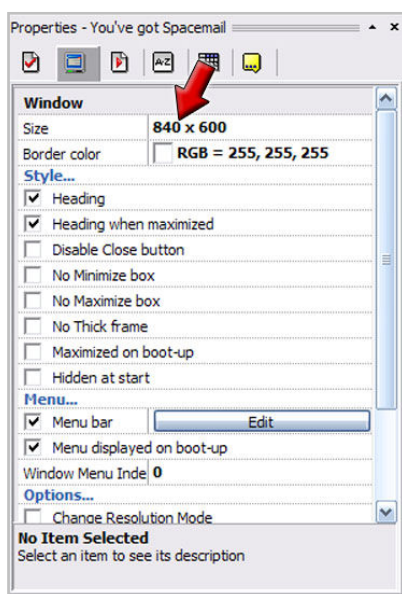
- ☐ If you have any problems with this tutorial, or notice that there are some mistakes present, please, contact me and I'll do my best to help you and replace all the errors with correct information.

Contact me at: marchewkowy@gmail.com

- ☐ **Note:** I've been receiving some reports that not all e-mails get to me for some reason. Seems that some of them (quite a lot) end up in my spambox or are blocked out by the server. I dunno why this is happening, so if you're experiencing any difficulties with delivering me a message or haven't received a reply in quite some time, please, send me another e-mail at marchewkowy@wp.pl, making sure that its title begins with "To Koobare:". I'll do my best to check both these e-mails regularly.

Part I: Setting up the application

Time to get to your mailbot, **people are waiting for their Spacemail to be delivered!** I'll start off simple, by showing you how to establish the ground for our work – or, simply put, **how to create and set up our application**. If you already know how to do this and/or are tired of such a basic approach, just head to **Part II** of this tutorial (a few pages down). This section is meant for the rooks, people who are still more on the "beginner" level than the "beginner intermediate".



Anyways, here we go... Open Multimedia Fusion 2, **create a new application** and save it onto your hard drive (it's always a good idea to have the *Autobackup* option of MMF2 turned on – check your MMF2's *Preferences*). Rename it to "**You've got Spacemail**". Now, go to your application's **Properties window** (if it didn't open up by itself, right click on your application's name in the workspace toolbar and select *Properties* from the drop-down menu), and select the **Window** tab (second from the left). Set the window size to **840x600**. If MMF2 asks you if you'd like to modify the size of the existing frame as well, select "Yes". If it doesn't ask you for itself, just open up the

Storyboard Editor and manually change the size of the frame to **840x600**. If you wish, you may easily brand our game – return to the **Properties window** of your application (you may need to click the name of your application in the *Workspace toolbar*), select the **About** tab (first from the right), and change the “Description” field to “You’ve got Spacemail! – The Tutorial”.

Once you’re done with that, open up the **Values** tab (third one from the right, the one with the “A-Z” symbol), which will enable us to edit the **Global Values** for our application. We actually need just one GV: create it and name it “**Delivered**”, making sure that it’s set to **zero**. This will prove useful a bit later on. Got it? Great! That means that it’s time for **Part II**. Move on!



- Interested in one game type in particular? Would like to learn about something that hasn't been covered in any of the released tutorials yet? Got an idea that could interest other tutorial-readers? Just drop me an e-mail!

Contact me at: marchewkowy@gmail.com

or write to the auxiliary address: marchewkowy@wp.pl

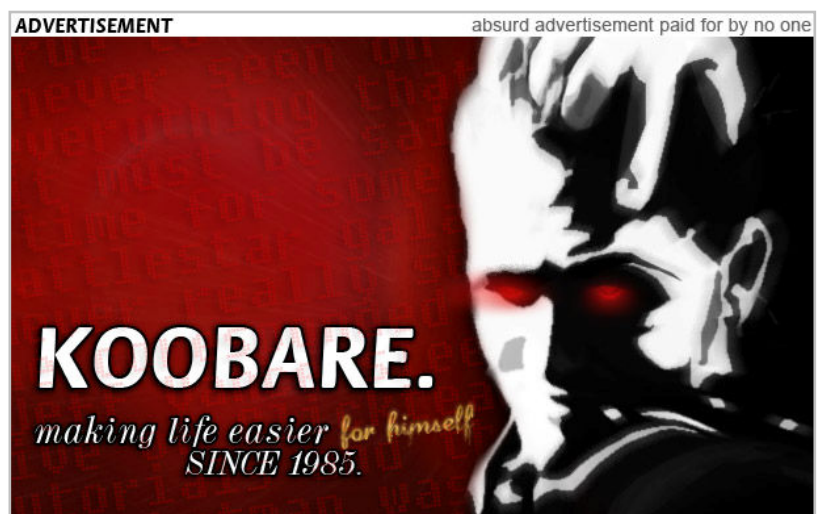
Part II: Making your life a bit easier...

Since my third tutorial I've **always been facing a dilemma** while writing the **part two** of any tut – traditionally the section in which we get hold of all the objects needed (not always, though, sometimes this happens in part three or even four, depending on the layout of that particular document). This ever-occurring dilemma goes like this: should I do everything – and I mean **EVERYTHING** – step-by-step, in a “I'll take your hand and take you with me” kind of thing, or not? Should I make sure that you – the readers, the users, people who I write for – take each and every object on the list and create all of them by yourself, placing them into the right positions, setting up all the preferences, fade-in effects, etc... Or should I rather not bore you to death, since you had to do this three times in the last two weeks already and you sure as heck know every little detail about placing a new background or an active object into the frame?

As these tutorials evolved (by the way, have you noticed the huge graphical progress that we've made

since “Smelly Claw”?), so did my solutions to this ever-present dilemma. At first I decided to let you just copy all the objects from a previously prepared application – just a simple copy 'n' paste operation and we were home. Recently, I've come up with a more logical solution – after the introduction and the optional “setting up the application” part we're simply going to open up that pre-prepped app and use it as the base for our project. This solution will now be applied in all *beginner-intermediate* and *intermediate* level tutorials, whereas tuts for novices will be treated with a “hands-on all the away” approach.

Since we're now clear about this, let's not waste any more time on organizational issues... There were quite a lot of them in this doc already, since I'm reorganizing everything to match your suggestions and my own modernizing concepts... But that's it, I promise. To continue, just close what you've done so far (if you actually didn't just skip section I of this tut) and open up the **you've-got-spacemail-start.mfa** file (should be in the same directory as this tutorial, since they were packed together into the same archive). And then it's time for...

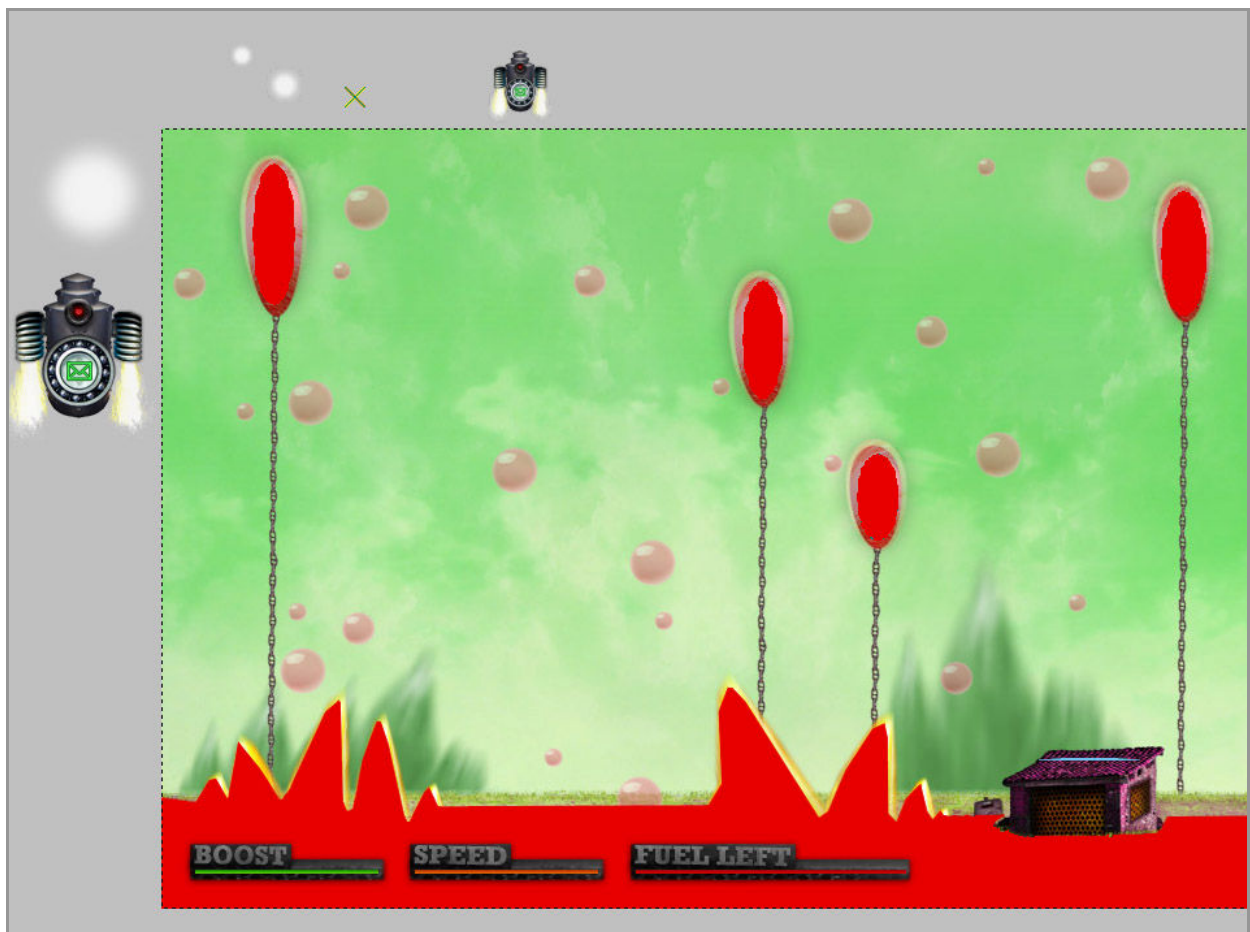


Please note: most of the objects use alpha channels, a feature that is unavailable in Games Factory 2 (TGF2 users should use basic library objects or create their own graphics instead – I generally would advise you to upgrade to MMF2 as soon as possible, since you’re missing out on some really good stuff, a lot of quite impressive features).

Part III: Knowing what’s what.

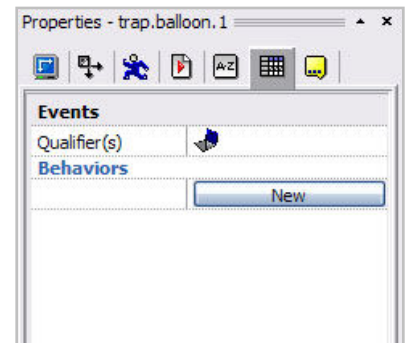
Once you’ve opened our basis app, open up the first and only frame (it should be entitled “**Spacemail**”) and take a look around. There’s a bunch of objects laying around, just waiting for you to give them a purpose, to script them all together into a nice, good-playin’ game.

You should be looking at something like this right now:



Notice the **red-colored “detectors”** that cover a big chunk of the background... Not looking very well, are they? Not really blending in with the rest of the objects? Don't be worried, they won't be seen in the game, as we will make them invisible. They are here to help us detect the moment in which the mailbot crashes into one of the balloons or into the ground.

Now, choose one at random and check out the “Events” tab in its properties window (it's the second tab from the right)... Notice that they are all assigned to the **“Traps” qualifier group**, which will enable us to easily create events that control all five of them at the same moment, with just a single line of code. This will prove useful (saving us some time and making our code better organized, cleaner) a bit later on, in the scripting part of this tutorial.



Another thing to notice here: there are two mailbots present outside the frame. The first one to the left is a bit bigger, with a simple path movement set in its preferences – this is the “intro” mailbot. The second one, more to the right and top corner of the screen, is the actual mailbot that will be used during the gameplay – it has a preconfigured “Bouncing Ball” movement, with the “initial direction” pointing down (direction number 24). Of course, it won't be behaving like a bouncing ball once we're done with it – this movement is just a great basis for creating your own, custom ones.



Time to check out all those little details of all those little objects laying around the frame... Below you can find an alphabetical list of all the objects, with their short description, their purpose and some characteristic properties all written down... Enjoy:

Object's name:	So... What is it?
backdrop	<p><u>Just a simple backdrop object – the background for our game.</u></p> <p>WHY IT'S HERE?</p> <p>Well, we need some kind of a background, don't we...? This is a picture that I've put together for this tut, it shows the bizarre world of Voopookoo, a distant planet colonized a few years ago by fearless pioneers from Australian Outback (somehow they didn't mind living amongst deserts and sun-burned canyons full of snakes and giant spiders). One of these pioneers awaits news about his family on Earth – it's imperative that we deliver him his messages as soon as possible!</p>

boost.counter	<p><u>A horizontal bar counter, used to control our mailbot's boost.</u></p> <p>PROPERTIES</p> <p>It is set as a horizontal bar counter, counting from left, with a vertical gradient fill type (set to two different shades of green). Its Initial and Minimum Values are both set to 0, its Maximum Value is set to 50.</p> <p>WHY IT'S HERE?</p> <p>Because we need some way of controlling our maildroid's engine boost.</p> <p>HOW DOES IT WORK?</p> <p>Well, it's a pretty simple system, really... During the time that the "Up Arrow" button is being pushed, "1" is added to this counter on every loop. Furthermore, when that condition is met, the current value of this counter is subtracted from the "fuel.counter" object's value. When the "Up Arrow" button is NOT being pressed, "1" is subtracted from the current value. If the "Down Arrow" button is being pressed, we subtract "3".</p>
engine.smoke and engine.smoke 2	<p><u>Two active objects acting as smoke puffs from our mailbot's engines.</u></p> <p>TRANSITIONS</p> <p>Both "engine.smoke" objects have their fade-out transitions already set for you – the first one is set to "Zoom" (at 0.56 of a second), the second one is set to "Fade" (set to a tiny duration of 0.22 of a second).</p> <p>WHY IT'S HERE?</p> <p>Thanks to my usual smoke-related trick (create a "smokey" object with a nice fade-out transition and destroy it immediately) we'll have a nice visual effect around our mailbot's engines.</p>
explosion	<p><u>Active object used as our explosion blast.</u></p> <p>INK EFFECT</p> <p>The Ink Effect of this object has been set to Add. This will give us a nice visual effect of a blindingly white explosion.</p> <p>TRANSITIONS</p> <p>The fade-out transition of this object is set to "Zoom" (0.34 sec.).</p> <p>WHY IT'S HERE?</p> <p>'Cause we needed an explosion – a visual effect that will appear when our mailbot hits one of the balloons or falls onto the ground (in fact hitting one of the red-colored "detectors" that we talked about a bit sooner).</p>
fuel.counter	<p><u>A horizontal bar counter, used to control our mailbot's fuel reserves.</u></p> <p>PROPERTIES</p> <p>It is set as a horizontal bar counter, counting from left, with a vertical gradient fill type (set to two different shades of red). Its Initial and Maximum Values are both set to 35000, its Minimum Value is set to 0.</p> <p>WHY IT'S HERE?</p> <p>Because this whole game is about fuel: if you use up too much, you won't be able to land and you'll crash miserably.</p>

	<p>HOW DOES IT WORK?</p> <p><i>This will look a lot easier in the Event Editor than on paper (or on the PDF's pages, in fact), but here it goes: on every loop subtract "1" from this counter, when the "Up Arrow" is pressed, subtract the value of the "boost.counter" object, if the left or right arrow keys are pressed, subtract "2". If the mailbot object crashes into a small bubble, subtract "800". If it crashed into a medium-sized bubble – subtract "1000". If it was a big bubble – subtract "1500".</i></p>
killer.bubble.big	<p><u><i>A big, pink floating bubble that eats up our mailbot's fuel reserves.</i></u></p> <p>TRANSITIONS</p> <p><i>The fade-out and fade-in of this object are both set to "Zoom" (0.79 sec).</i></p> <p>MOVEMENT</p> <p><i>The movement is set to "Bouncing Ball", with Speed set at "5", Deceleration at "0", the "Moving at start" option being turned on.</i></p> <p>HOW DOES IT WORK?</p> <p><i>Well, it's pretty simple: once the mailbot crashes into this bubble, it breaks apart with a "pop!" sound, eats up some fuel and then reappears in a random position on the screen. Furthermore, from time to time a randomly selected "killer.bubble.big" object suddenly changes its direction, without any previous notice...</i></p>
killer.bubble.small and killer.bubble.medium	<p><u><i>Smaller versions of the "killer.bubble.big" object.</i></u></p> <p>WHY IT'S HERE?</p> <p><i>Well, they have exactly the same purpose as their bigger brother, "killer.bubble.big" – they're here to eat up the player's fuel...</i></p> <p>MOVEMENT & TRANSITIONS</p> <p><i>They're basically the same as in the "killer.bubble.big" object, the only difference is that the "killer.bubble.small" has a shorter duration set for the "Zoom" fade-out (0.56 of a second) and fade-in (0.67) transitions.</i></p>

landing.zone	<p><u>The landing zone for our mailbot (made invisible at the start of the frame).</u></p> <p>WHY IT'S HERE?</p> <p>A small active object that is used as the “landing zone” for our mailbot – our droid will land only when it is in collision with this object (it also needs to have the proper speed and the space bar must be pushed in the right moment). It is placed on the pink spacecabin’s roof.</p>
randomizer	<p><u>A small active object that changes its position at random on every loop.</u></p> <p>WHY IT'S HERE?</p> <p>This object helps us to randomize the location at which a new bubble is created once it has ruptured during a collision with our mailbot. Its X position is chosen randomly from the range of 0 to 830, whereas its Y position is set to something from the area of 0 to 590.</p>
mailbot.game	<p><u>Our main protagonist – Mailbot 7890-Kappa-91, also known as Raymond.</u></p> <p>TRANSITIONS</p> <p>The fade-out transition of this object is set to “Zoom” (1.58 sec.).</p> <p>MOVEMENT</p> <p>The movement is set to “Bouncing Ball”, with Speed set to “30”, Deceleration at “0”, the “Moving at start” option being turned off, whereas the Initial direction being set to 24 (down).</p> <p>WHY IT'S HERE?</p> <p>Well, this is the hero of our game – where else should he be? We control Raymond with the help of the Arrow keys. To land him on the roof of the spacecabin, just get him close enough (and that means really, really close), slow down his fall and then press space.</p>
mailbot.intro	<p><u>The mailbot used in our short “intro”.</u></p> <p>WHY IT'S HERE?</p> <p>We’ve already learned everything there is to learn about this object. If you require more information, just “reverse engineer” it, by checking out its movement settings and animation frames.</p>
sign.boost and sign.fuel and sign.speed	<p><u>Three little active objects that act as the background for our counters.</u></p> <p>WHY IT'S HERE?</p> <p>Well, to be honest, they’re here just to make those counters look better... Oh, and they have names on them too, so I guess that “informational purposes” count as well.</p>
speed.counter	<p><u>A horizontal bar counter, used to display our mailbot’s speed.</u></p> <p>PROPERTIES</p> <p>It is set as a horizontal bar counter, counting from left, with a vertical gradient fill type (set to orange and brown). Its Initial and Minimum Values are both set to 0, its Maximum Value is set to 35.</p>

	<p>WHY IT'S HERE?</p> <p><i>Well, because we really need to give the player some tips on how fast his mailbot is going...</i></p> <p>HOW DOES IT WORK?</p> <p><i>This counter is less important for the overall game engine than the last two were – but still, it's crucial if we want Raymond's expedition to have a happy ending. This counter is checked for its value during the landing – and it is always set to have the same speed as the "mailbot.game" object.</i></p>
<p>trap.ground and trap.balloon.1 and trap.balloon.2 and trap.balloon.3 and trap.balloon.4</p>	<p><u>Five "trap" objects – the red, invisible collision detectors.</u></p> <p>PROPERTIES</p> <p><i>All of these objects share a common qualifier – they all belong to the group entitled "Traps".</i></p> <p>WHY IT'S HERE?</p> <p><i>So that we know when does the collision between the mailbot and the ground (or the chained balloons) take place.</i></p> <p>HOW DOES IT WORK?</p> <p><i>Nothing fancy here: all the "traps" are hidden at the start of the frame, but even when they are invisible, a collision with any of them will bring our mailbot down, smashing it to pieces.</i></p>

The invisible detectors

A nice thing to remember for the future, to use in your own projects, especially those with a giant bitmap background: using invisible detectors instead of slicing your background into separate objects can sometimes prove quite useful and save you a lot of time.

Part IV: Time for a bit of programming.

It's time for my favorite part! Save your project (always remember to save it from time to time, that's a must!) and open the **Event Editor**. If you're new to my tutorials, let me introduce you to the event-recording system that I use. If you know it already – just skip this frame below and quickly move on to the coding part:

Koobare's MMF-to-paper coding system

IF (Condition): [Object for the condition] > Condition group > Condition

THEN (Action): [Object for the action] > Action group > Action

Seems simple, right? Well, that's just because IT IS simple. All the conditions are marked in red, while actions are written in fancy blue.

Object names are always put in [square brackets]. The final condition/action is always in *Italic*. If we'll have a multi-condition event, then it'll be like this:

IF (Condition 1): [Object for condition 1] > Condition group 1 > Condition 1

IF (Condition 2): [Object for condition 2] > Condition group 2 > Condition 2

THEN (Action): [Object for the action] > Action group > Action

Whereas a multi-action event looks like this:

IF (Condition): [Object for condition] > Condition group > Condition

THEN (Action 1): [Object for the action 1] > Action group 1 > Action 1

THEN (Action 2): [Object for the action 2] > Action group 2 > Action 2

If you'll have to input anything by keyboard, it will be indicated by coloring the text green and using < angle brackets >, like this (this marking will be soon obsolete):

< Set the Global Value A to 32 >

Additional comments, instructions and info will be put in << double angle brackets >>, using a different color (this marking will soon be also used for input):

<< Select any wave sound from the MMF2's sound library >>

From time to time I'll also use this style to throw in some extra tips and tricks about MMF2 and more advanced coding techniques. All you have to do is to go step-by-step through all the listed events and keep one eye on your Event Editor, and the second one on this tutorial...

Let's deliver some Spacemail, already!

1) Firstly, let's start off with the conventional **"Start of frame"** event, which I usually create at the very beginning of the events list. This event – triggered when someone starts our game –

will make quite a lot of objects invisible (all the traps, all the counters, all the “sign” objects, the landing zone and the randomizer – the “signs” and counters will be returned to their visible status once our little “intro” has ended) and, in addition, will play two sample files (search for the “Warm up.wav” sound on your MMF2 Bonus Materials disc, whereas the “flyjet-cut.wav” file is supplied in the same archive as this tutorial):

```
IF: [Storyboard Controls] > Start of frame
THEN: [Group.Traps] > Visibility > Make invisible
THEN: [randomizer] > Visibility > Make invisible
THEN: [landing.zone] > Visibility > Make invisible
THEN: [speed.counter] > Visibility > Make invisible
THEN: [fuel.counter] > Visibility > Make invisible
THEN: [boost.counter] > Visibility > Make invisible
THEN: [sign.fuel] > Visibility > Make invisible
THEN: [sign.speed] > Visibility > Make invisible
THEN: [sign.boost] > Visibility > Make invisible
THEN: [Sound Object] > Samples > Play and loop sample
<< Choose the “Warm Up.wav” sound, loop it 0 times, which means infinitely >>
THEN: [Sound Object] > Samples > Play and loop sample
<< Choose the “flyjet-cut.wav” sound, loop it 0 times, which means infinitely >>
```

And with just a bit of MMF2 magic – we’ve got our first event up and ready! Ain’t it great?

Now, before we march onto the next event, let’s set up all the **Event Groups** that we’ll need in our event list... Firstly, create three groups beneath the first event and call them “**Intro**”, “**Spacemail**” and “**Landed**” respectively. Make sure that both “Intro” and “Landed” have the “**Active when frame starts**” option turned **ON**, whereas “Spacemail” should have this option turned **OFF**. Once that’s done, open up the “Spacemail” group and create two more subgroups inside of it: “**Engines**” and “**Killer Bubbles**” (both should be set active when frame starts). And just when you thought we were done with all of this grouping... let’s create another subgroup! Create it inside the “Engines” group and name it “**Engine Smoke**” (once again – let it be active when frame starts)... Got it? Great! We can now finally move on to some more scripting!

2) Time for our event *numero duo*! This one ends our “intro” stage (which will last for a whole 5 seconds, folks) enabling all those “sign” objects and counters to reappear once again... This little thingie will also destroy the “mailbot.intro” object, play around with group activation and start off two sound samples... Just follow my lead, and we’ll be there in no time:

IF: [The Timer Object] > Is the timer equal to a certain value?

<< Set the timer to 5 seconds >>

THEN: [mailbot.intro] > Destroy

THEN: [Special Object] > Group of events > Activate

<< Select the Spacemail group >>

THEN: [speed.counter] > Visibility > Make object reappear

THEN: [fuel.counter] > Visibility > Make object reappear

THEN: [boost.counter] > Visibility > Make object reappear

THEN: [sign.fuel] > Visibility > Make object reappear

THEN: [sign.speed] > Visibility > Make object reappear

THEN: [sign.boost] > Visibility > Make object reappear

THEN: [Sound Object] > Samples > Play sample

<< Choose the "PULSE07.wav" sound >>

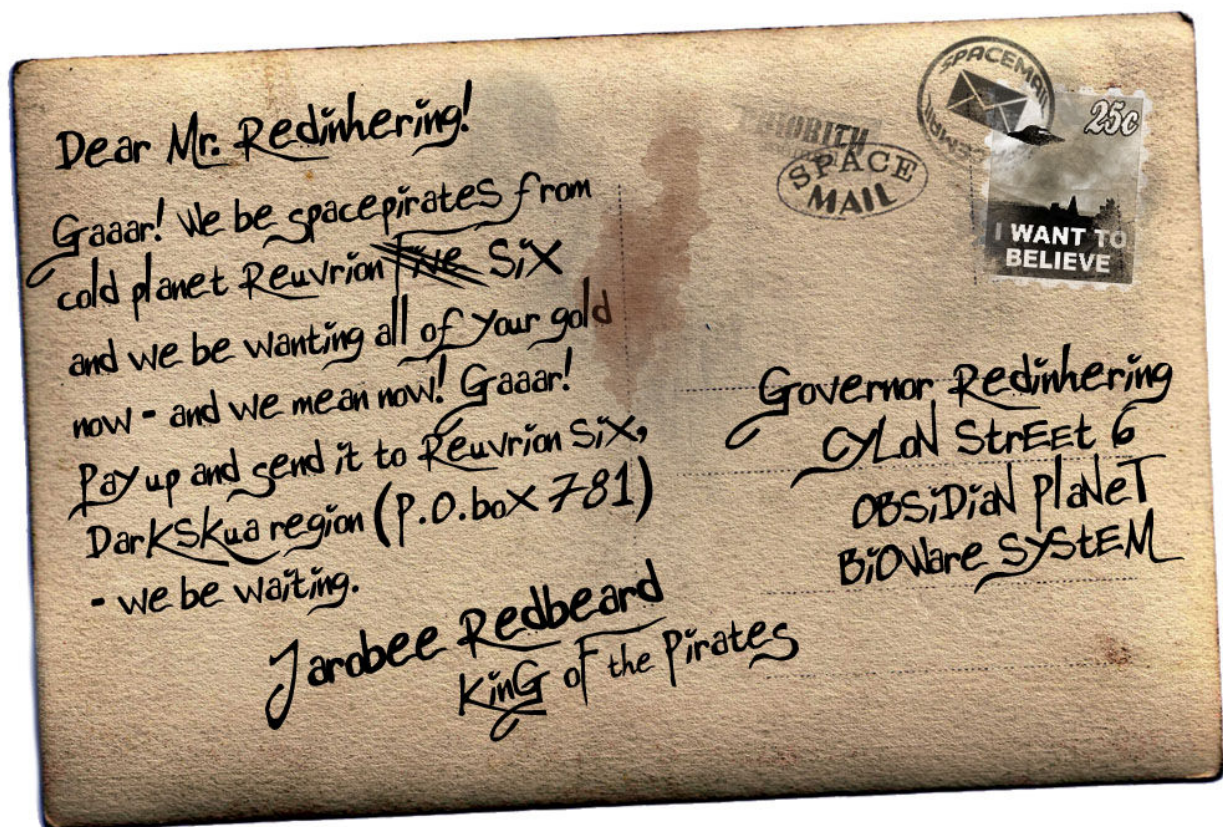
THEN: [Sound Object] > Samples > Stop a specific sample

<< Choose the "flyjet-cut.wav" sound >>

THEN: [Special Object] > Group of events > Deactivate

<< Select the Intro group >>

Got it? Well then, good for you! Drag and drop this event to the second position in the events list (just for neatness' sake), just above the "Intro" group...



3) Now its time to work on that little 5-second-long “intro” of ours... Create all these events inside the **“Intro”** event group, one after another:

IF: [The Timer Object] > Every

<< Set the timer to 0.17 of a second >>

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

```
<< Set the coordinates to x=-38, y=75, relative to [mailbot.intro] object >>
```

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

<< Set the coordinates to x=38, y=76, relative to [mailbot.intro] object >>

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

<< Set the coordinates to x=44, y=58, relative to [mailbot.intro] object >>

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

<< Set the coordinates to x=-47, y=59, relative to [mailbot.intro] object >>

Aaaand here's the second one, remember to put these in the "Intro" group:

IF: [The Timer Object] > Every

<< Set the timer to 0.21 of a second >>

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

```
<< Set the coordinates to x=-48, y=74, relative to [mailbot.intro] object >>
```

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

```
<< Set the coordinates to x=43, y=71, relative to [mailbot.intro] object >>
```

THEN: [Create New Objects] > Create Object


<< Select the [engine.smoke 2] object >>

<< Set the coordinates to x=30, y=59, relative to [mailbot.intro] object >>

THEN: [Create New Objects] > Create Object

<< Select the [engine.smoke 2] object >>

```
<< Set the coordinates to x=-31, y=51, relative to [mailbot.intro] object >>
```

All the events All the objects																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
1	• Start of Frame	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																</

Same drill, same group, almost identical event...

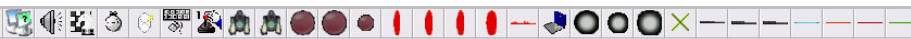

IF: [The Timer Object] > Every
 << Set the timer to 0.25 of a second >>
THEN: [Create New Objects] > Create Object
 << Select the [engine.smoke 2] object >>
 << Set the coordinates to x=-38, y=67, relative to [mailbot.intro] object >>
THEN: [Create New Objects] > Create Object
 << Select the [engine.smoke 2] object >>
 << Set the coordinates to x=38, y=64, relative to [mailbot.intro] object >>

Here's something a bit different – two last events that go into the “Intro” event group...

IF: [Special Object] > Always
THEN: [engine.smoke 2] > Animation > Change > Animation Sequence
 << select “Smoke” >>

IF: [engine.smoke 2] > Movement > Is stopped?
THEN: [engine.smoke 2] > Destroy

And that concludes our efforts when it comes to the “Intro” group... You can take a look at this visual aid to see how it looks in my Event Editor...

All the events All the objects																												
1	• Start of Frame	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>															<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	• Timer equals 05"-00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>						<input checked="" type="checkbox"/>													<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	Intro																											
4	• Every 00"-17					<input checked="" type="checkbox"/>																						
5	• Every 00"-21					<input checked="" type="checkbox"/>																						
6	• Every 00"-25					<input checked="" type="checkbox"/>																						
7	• Always																											
8	•  is stopped																					<input checked="" type="checkbox"/>						
9	• New condition																					<input checked="" type="checkbox"/>						
10	Spacemail																											

4) Let's continue... Open up the “Spacemail” group (have you remembered to make this group inactive when frame starts?) and create this little event:

IF: [Special Object] > Group of events > On group activation
THEN: [mailbot.game] > Movement > Start

This little thingie ensures that our mailbot (the game one, not the intro one) will be moving once the actual game starts (when the timer hits 5 seconds and activates the “Spacemail” group).

5) Step five: another event that goes right into the “Spacemail” group... Notice that the value of the “speed.counter” object is always exact as the speed of the “mailbot.game” one. You can also observe how the “randomizer” is forced to jump to a random position on every loop:

```
IF: [Special Object] > Always
THEN: [fuel.counter] > Subtract from Counter
<< input: 1 >>
THEN: [speed.counter] > Set Counter
<< input: Speed( "mailbot.game" ) >>
THEN: [randomizer] > Position > Set X position
<< input: Random(830) >>
THEN: [randomizer] > Position > Set Y position
<< input: Random(590) >>
THEN: [engine.smoke] > Destroy
```

6) Yet another simple thingie that goes into the “Spacemail” event directory...

```
IF: [mailbot.game] > Position > Test position
<< Select “Leaves in the top?” – arrow leaving the frame at the top >>
<< Select “Leaves in the right?” – arrow leaving the frame to the right >>
<< Select “Leaves in the bottom?” – arrow leaving the frame at the bottom >>
<< Select “Leaves in the left?” – arrow leaving the frame to the left >>
THEN: [mailbot.game] > Movement > Bounce
```

7) Let’s see what happens when our mailbot crashes into one of the collision detectors...

```
IF: [mailbot.game] > Collisions > Another object > [Group.Traps]
THEN: [Create New Objects] > Create Object
<< Select the [explosion] object >>
<< Set the coordinates to x=1, y=12, relative to [mailbot.game] object >>
THEN: [Sound Object] > Samples > Play sample
<< Choose the “EXPLOD03.wav” sound >>
THEN: [Sound Object] > Samples > Play sample
<< Choose the “CLATTER2.wav” sound >>
THEN: [mailbot.game] > Destroy
THEN: [Sound Object] > Samples > Stop a specific sample
<< Choose the “flyjet-cut.wav” sound >>
THEN: [Sound Object] > Samples > Stop a specific sample
<< Choose the “KNOCK.wav” sound >>
```

8) We're still in the "Spacemail" group, remember that – every event from step 4 up until now (and up until step 11) goes into that set... Here's a simple event for ya':

```
IF: [explosion] > Animation > Has an animation finished?  
<< select "Stopped" >>  
THEN: [explosion] > Destroy  
THEN: [Sound Object] > Samples > Play sample  
<< Choose the "Bash 2.wav" sound >>
```

9) Here's what happens when our mailbot has been destroyed...

```
IF: [mailbot.game] > Pick or count > Have all been destroyed?  
THEN: [Storyboard Controls] > Restart the current frame
```

10) Here's a bit longer event, that is commenced once the mailbot's fuel reserves are down to zero... Basically this makes our mailbot fall from the skies...

```
IF: [fuel.counter] > Compare the counter to a value  
<< compare whether it is Lower or Equal 0 >>  
THEN: [Special Object] > Group of events > Deactivate  
<< Select the Engines group >>  
THEN: [mailbot.game] > Direction > Select direction  
<< Select direction 24 (down) >>  
THEN: [speed.counter] > Visibility > Make invisible  
THEN: [fuel.counter] > Visibility > Make invisible  
THEN: [boost.counter] > Visibility > Make invisible  
THEN: [sign.fuel] > Visibility > Make invisible  
THEN: [sign.speed] > Visibility > Make invisible  
THEN: [sign.boost] > Visibility > Make invisible  
THEN: [mailbot.game] > Animation > Change > Animation Sequence  
<< Select "Falling" >>  
THEN: [mailbot.game] > Movement > Set Speed...  
<< Type this in: 50 >>  
THEN: [Sound Object] > Samples > Stop a specific sample  
<< Choose the "flyjet-cut.wav" sound >>
```

11) And here's the last event that's going to go directly into the "Spacemail" event group... This one is just a supplement to the previous one.

<< compare whether it is *Lower or Equal 0* >>

IF: [Special Object] > Limit conditions > *Run this event once*

THEN: [Sound Object] > Samples > *Play and loop sample*

<< Choose the “KNOCK.wav” sound, loop it 0 times, which means infinitely >>

THEN: [Sound Object] > Samples > *Play sample*

<< Choose the “Impact fx 1.wav” sound >>

You can once again compare your progress with mine, helps to this little visual aid:

[illegible]

Once you're done comparing, let's get back to business...



12) Now it's time to open up that **"Engines"** event group. Create all these events inside this set – they are all responsible for steering, for controlling the mailbot on runtime:

IF: **[boost.counter] > Compare the counter to a value**

<< compare whether it is *Greater or equal 30* >>

THEN: **[mailbot.game] > Direction > Select direction**

<< Select direction *8 (up)* >>

THEN: **[mailbot.game] > Movement > Set Speed...**

<< input: *0-(30-value("boost.counter"))* >>

IF: **[boost.counter] > Compare the counter to a value**

<< compare whether it is *Lower than 30* >>

THEN: **[mailbot.game] > Direction > Select direction**

<< Select direction *24 (down)* >>

THEN: **[mailbot.game] > Movement > Set Speed...**

<< input: *35-value("boost.counter")* >>

IF: **[Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed**

<< press *UP ARROW* on your keyboard >>

THEN: **[boost.counter] > Add to Counter**

<< input: *1* >>

THEN: **[fuel.counter] > Subtract from Counter**

<< input: *value("boost.counter")* >>

Here's an event with a *negated* condition. To negate a condition just right-click on it and choose "negate" from the drop-down menu:

[NEGATE] IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed

<< press *UP ARROW* on your keyboard >>

THEN: **[boost.counter] > Subtract from Counter**

<< input: *1* >>

IF: **[Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed**

<< press *DOWN ARROW* on your keyboard >>

THEN: **[boost.counter] > Subtract from Counter**

<< input: *3* >>

Now it's the time for a series of events that will help us to control the sideways movement of Mailbot 7890-Kappa-91 (which can prove pretty useful when navigating, don't you think?):

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press LEFT ARROW on your keyboard >>
IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press UP ARROW on your keyboard >>
THEN: [mailbot.game] > Direction > Select direction
<< Select direction 12 (top-left) >>
THEN: [fuel.counter] > Subtract from Counter
<< input: 2 >>

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press RIGHT ARROW on your keyboard >>
IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press UP ARROW on your keyboard >>
THEN: [mailbot.game] > Direction > Select direction
<< Select direction 4 (top-right) >>
THEN: [fuel.counter] > Subtract from Counter
<< input: 2 >>

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press LEFT ARROW on your keyboard >>
[NEGATE] IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press UP ARROW on your keyboard >>
THEN: [mailbot.game] > Direction > Select direction
<< Select direction 22 (down-left) >>
THEN: [fuel.counter] > Subtract from Counter
<< input: 2 >>

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press RIGHT ARROW on your keyboard >>
[NEGATE] IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
<< press UP ARROW on your keyboard >>
THEN: [mailbot.game] > Direction > Select direction
<< Select direction 26 (down-right) >>
THEN: [fuel.counter] > Subtract from Counter
<< input: 2 >>

Thanks to these little four events our mailbot will be able to maneuver around those crazy chained balloons and deadly fuel-eating bubbles... Good work! Don't get too relaxed, though, there's still a lot to do... These two events will control the sound of the mailbot's engine:

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
 << press UP ARROW on your keyboard >>
 IF: [Special Object]>> Limit conditions>> Only one action when event loops
 IF: [fuel.counter] > Compare the counter to a value
 << compare whether it is Greater than 1 >>
 THEN: [Sound Object] > Samples > Play and loop sample
 << Choose the “flyjet-cut.wav” sound, loop it 0 times, which means infinitely >>

[NEGATE] IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
 << press UP ARROW on your keyboard >>
 IF: [Special Object]>> Limit conditions>> Only one action when event loops
 THEN: [Sound Object] > Samples > Stop a specific sample
 << Choose the “flyjet-cut.wav” sound >>

13) And that’s all when it comes to the “Engines” group... Let’s move on to the next event set – the “Engine smoke” one. Open it up and create these two events inside (both are responsible for creating our cool smoke-like visual effect):

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
 << press UP ARROW on your keyboard >>
 IF: [The Timer Object]>> Every
 << Set the timer to 0.14 of a second >>
 IF: [fuel.counter] > Compare the counter to a value
 << compare whether it is Greater than 1 >>
 THEN: [Create New Objects] > Create Object
 << Select the [engine.smoke] object >>
 << Set the coordinates to x=-15, y=29, relative to [mailbot.game] object >>
 THEN: [Create New Objects] > Create Object
 << Select the [engine.smoke] object >>
 << Set the coordinates to x=17, y=31, relative to [mailbot.game] object >>

IF: [Keyboard & Mouse Object] > The Keyboard > Repeat while key is pressed
 << press UP ARROW on your keyboard >>
 IF: [The Timer Object]>> Every
 << Set the timer to 0.34 of a second >>
 IF: [fuel.counter] > Compare the counter to a value
 << compare whether it is Greater than 1 >>
 THEN: [Create New Objects] > Create Object
 << Select the [engine.smoke] object >>
 << Set the coordinates to x=-19, y=34, relative to [mailbot.game] object >>


```
<< Set the coordinates to x=19, y=34, relative to [mailbot.game] object >>
```

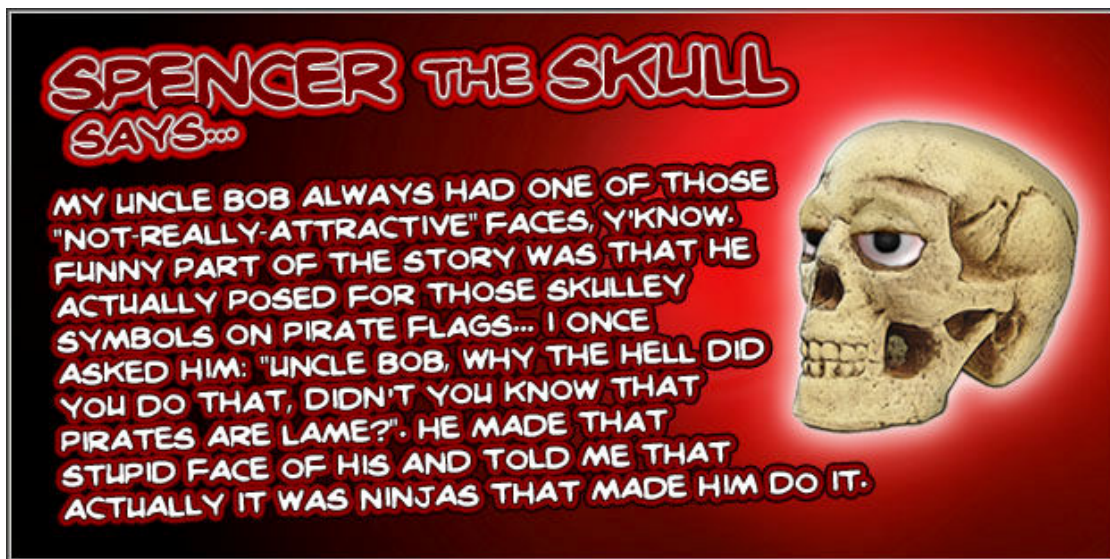
[illegible]

<< Choose the “POP1.wav” sound >>

<< Choose the “POP1.wav” sound >>

[illegible]

IF: [mailbot.game] >> Collisions >> Another object >> [killer.bubble.small]
 THEN: [killer.bubble.small] >> Destroy
 THEN: [Create New Objects] > Create Object
 << Select the [killer.bubble.small] object >>
 << Set the coordinates to x=0, y=0, relative to [randomizer] object >>
 THEN: [fuel.counter] > Subtract from Counter
 << input: 800 >>
 THEN: [Sound Object] > Samples > Play sample
 << Choose the "POP1.wav" sound >>



If any of the killer bubbles leaves the play area (they're floating randomly in all directions, so that's gonna' happen a lot), they are wrapped around it (appear from the other side):

IF: [killer.bubble.big] >> Position >> Test position
 << Select "Leaves in the top?" – arrow leaving the frame at the top >>
 << Select "Leaves in the right?" – arrow leaving the frame to the right >>
 << Select "Leaves in the bottom?" – arrow leaving the frame at the bottom >>
 << Select "Leaves in the left?" – arrow leaving the frame to the left >>
 THEN: [killer.bubble.big] >> Movement >> Wrap around play area

IF: [killer.bubble.medium] >> Position >> Test position
 << Select "Leaves in the top?" – arrow leaving the frame at the top >>
 << Select "Leaves in the right?" – arrow leaving the frame to the right >>
 << Select "Leaves in the bottom?" – arrow leaving the frame at the bottom >>
 << Select "Leaves in the left?" – arrow leaving the frame to the left >>
 THEN: [killer.bubble.medium] >> Movement >> Wrap around play area

```

IF: [killer.bubble.small] >> Position >> Test position
<< Select "Leaves in the top?" – arrow leaving the frame at the top >>
<< Select "Leaves in the right?" – arrow leaving the frame to the right >>
<< Select "Leaves in the bottom?" – arrow leaving the frame at the bottom >>
<< Select "Leaves in the left?" – arrow leaving the frame to the left >>
THEN: [killer.bubble.small] >> Movement >> Wrap around play area

```

From time to time one of the bubbles – chosen at random – suddenly bounces...

```

IF: [The Timer Object]>> Every
<< Set the timer to 1.00 second >>
IF: [killer.bubble.big] >> Pick or count >> Pick one at random
IF: [Special Object] >> X chances out of Y random
<< Input value: 1 >>
<< Input value: 7 >>
THEN: [killer.bubble.big] >> Movement >> Bounce

```

```

IF: [The Timer Object]>> Every
<< Set the timer to 1.00 second >>
IF: [killer.bubble.medium] >> Pick or count >> Pick one at random
IF: [Special Object] >> X chances out of Y random
<< Input value: 1 >>
<< Input value: 8 >>
THEN: [killer.bubble.medium] >> Movement >> Bounce

```

```

IF: [The Timer Object]>> Every
<< Set the timer to 1.00 second >>
IF: [killer.bubble.small] >> Pick or count >> Pick one at random
IF: [Special Object] >> X chances out of Y random
<< Input value: 1 >>
<< Input value: 12 >>
THEN: [killer.bubble.small] >> Movement >> Bounce

```

Got it? Great! That means that we are just **two events** from finally completing this tutorial! Woah, this was a big one, wasn't it? Anyway, we'll chit-chat a bit later – it's time to get to the finish line, to get all of this up and running!

14) Leave the "Killer Bubbles" group and go to the next one – the one entitled "**Landed**". These two events control what happens when the player actually manages to get our mailbot close enough to the spacecabin and to press space bar on time:

And here's the last event that are gonna' create today!

[illegible]

Page 30/32

This was a pretty long tutorial, but we finally made it! Hope that you'll find out that the game we have produced together was worth all this amount of work and time. Mailbot 7890-Kappa-91 finally has his shot at delivering the messages to the settlers of Voopookoo, and you – finally – have a chance to show that you're the best mailbot operator in the whole Galaxy Postage Office! Go out there and make us proud!

Thanks for your time and see you again soon!

Cheers!

Koobare

*If you have any questions, suggestions or just need help –
mail me at marchewkowy@gmail.com*

To all you pirate-lovers out there: don't get offended by my anti-pirate jokes, they are just jokes after all! Having said that... I truly, truly think that a ninja warrior could wipe the floor with a pirate, anytime, any day! ;) And no, I won't walk the plank, thanks for your proposal. ;)

All copyrighted materials, names, titles, images and visualizations (as "Battlestar Galactica", the xenomorph from "Alien", "The Hulk" etc.) belong to their respective owners and are used here exclusively as a parody or a satirist fan tribute – no copyright infringement intended.

You have been reading...



brought to you by
Koobare

Clickteam's funkiest
~~mercenary~~
mailbot operator



Created for Multimedia Fusion 2 & Multimedia Fusion 2: Developer

Always be sure to have your MMF2 up-to-date!