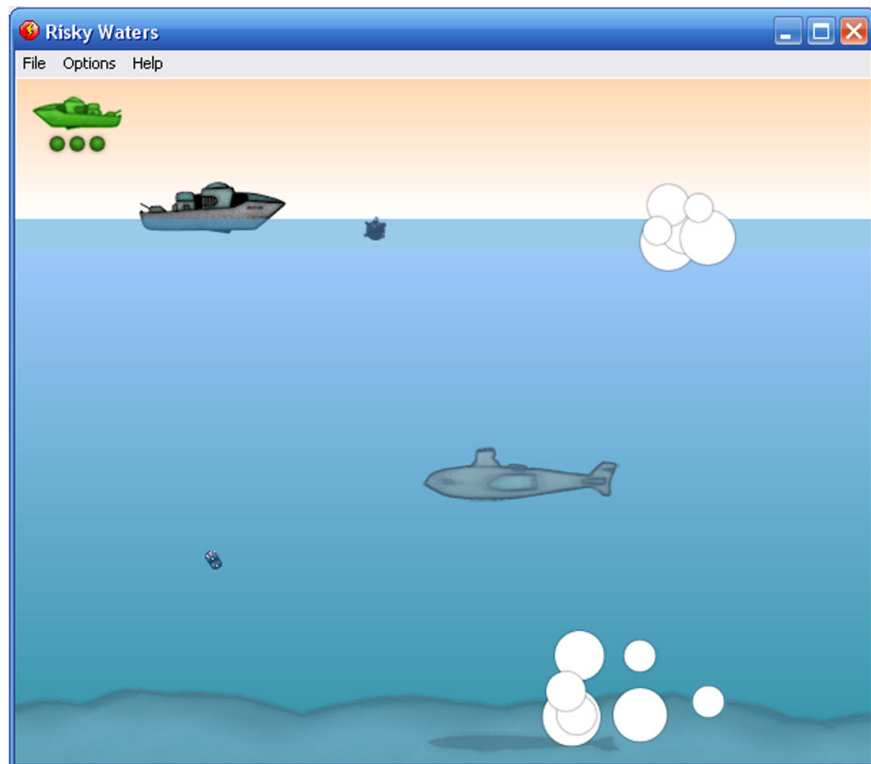


RISKY WATERS

a Multimedia Fusion 2 tutorial



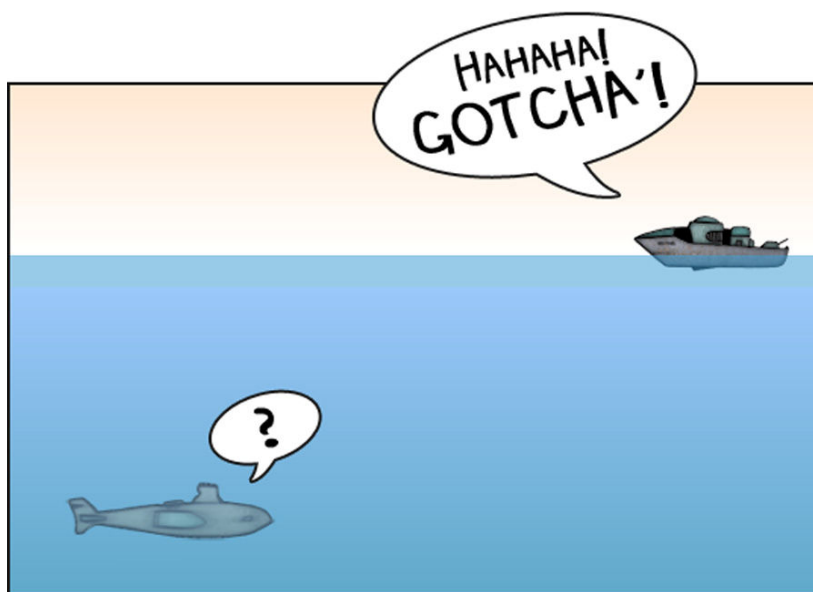
You may not use this tutorial for any other purpose than learning, working or having fun... In other words: You can use this PDF tutorial for anything You'd like, as long as it doesn't involve both a hammer and a squirrel.

Koobare
marchewkowy@gmail.com

Hi there, all!

Welcome to another one of Koobare's little tutorials (been a while...), teaching you how to effectively and efficiently use the best multimedia authoring tool ever - [Multimedia Fusion 2](#) by Clickteam! This tutorial is meant for intermediate beginners, people who are still a bit new to the fascinating world of MMF2 – if you're a bit more experienced Clicker, just skip this tutorial and take a look at the other ones that can be found on Clickteam's website... Unless, of course, you don't mind a bit of repetition and a few things that can be *old news* for ya'.

We'll start off simple, introducing you to the basic features of MMF2... But believe me: you'll soon find out that with basic features alone Multimedia Fusion 2 is the best tool in it's category. A simple "Sink the Sub!" game for now, but there's plenty ahead of you – adventure and roleplaying games, advanced platformers, turn-based tactics, multimedia presentations, screensavers and... Well, let's skip the listing part, I don't have enough time on my hands. Just believe me when I tell you this: you can do ANYTHING with MMF2, you just need a clear vision of what you're trying to accomplish and a bit of experience to get there. And – as far as experience is concerned – that's exactly what you need this tutorial for, right?



So, no more motivational chit-chat for now – let's gather up, listen up and create a plan! Our goal? To create "Risky Waters", a fun-to-play arcade game, with exactly 23 events coded in the Event Editor. The storyline for our game is pretty simple, but this shouldn't bother you too much – this is supposed to be an arcade game after

all. Our player will become a captain of *Sub Hunter 3000*, a super-advanced ship created solely for the purpose of... Well, hunting down subs. And that's it. No interstellar trips here, but I hope you weren't counting on any? Now, prepare your gear!

Torpedoes away, captain!

To sum it all up – here are some core features of the project we’re going to complete:

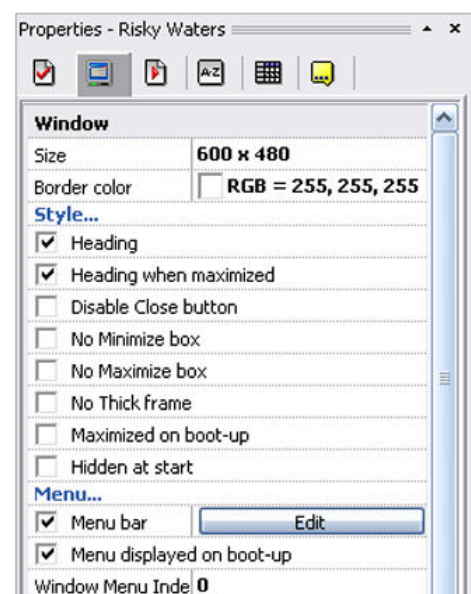
- We’re going to create a good-lookin’ arcade game about hunting down submarines.
- Our player will control a single ship – *Sub Hunter 3000* – capable of dropping depth charges into the ocean. Depth charges will explode on impact, but it will take a couple of them to sink a single submarine.
- The number of depth charges will be limited, but rechargeable.
- Enemy submarines won’t be helpless – they can release numerous submerging water-mines, which will explode on collision with either the player’s depth charges or the his ship. If they won’t hit anything on their way – they’ll float on the surface for a short time, before exploding.

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


Part I: Setting up the application.

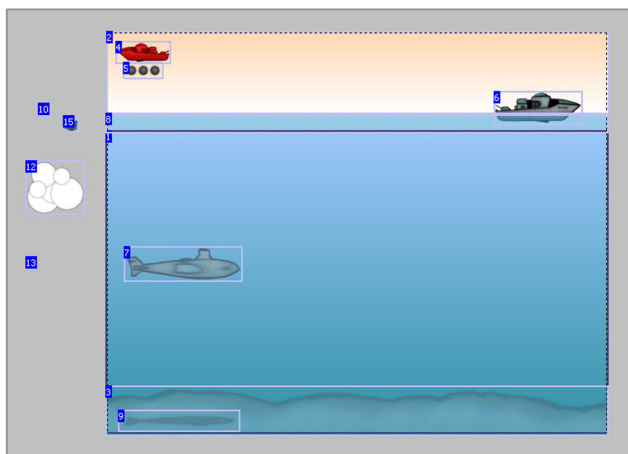
It’s time to have some fun! Open Multimedia Fusion 2, **create a new application** and save it onto your hard drive (it’s a pretty good thing to have the “Autobackup” option of MMF2 turned on – check your “Preferences”). Go to your application’s **Properties window** (if it didn’t open up by itself, 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 **600x480**. When that’s done, create your first frame (make it’s size identical to the size of our window: **600x480**) and continue to part two of this tutorial.



Part II: Importing objects

So, we're creating a game about a boat that drops depth charges into the ocean and tries to sink a submarine, right? So where the heck is that boat? And those depth charges? And the submarine? Huh, we don't even have anything that looks like an ocean, right?














Not a problem, though. I've got all those objects prepared for you – they are safely waiting in the “**riskywaterslibrary.mfa**” file, which was packed into the same archive as this .PDF tutorial. It's time to import them into our app! Firstly, make sure that you have the “*snap to grid*” option turned off (check the “view” menu and search for the  **Snap to Grid** button). Secondly, find the aforementioned “riskywaterslibrary.mfa” file and open it. Once it's loaded into your MMF2, open the first frame from the library file (actually, it should be the only frame in there). Got it? Great! Arrgh, matey, we've got it! 'Tis a treasure of sorts, for me aching ol' pirate eyes to see such beauty! Translation from Piratese to English: here they are, all shining and ready to use!



Now... Select all the objects available in the library's frame (press **CTRL+A** or select them by dragging a selection field over them), copy them into your application (CTRL+C, then CTRL+V) and place them correctly (this shouldn't be too tough if you've disabled the “*snap to grid*” option – take a look at the image to the left if you require visual guidance).

Please note: some 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).

Now, let's take a look at the objects we've just imported. There's not too many of them, but still enough to confuse you a bit if you don't know what's what. That's why I've created this neat little table for you – just take a look, read through it, and you'll have all the knowledge needed:

Icon:	Object:	So... What is it?
	ammo_counter	<i>A counter that shows how much ammo (depth charges) does the player still have. It renews over time.</i>
	boom	<i>A big explosion – an animation that is always created when something has blown up.</i>
	bottom_backdrop	<i>'Tis the bottom of the sea, lad! – a simple backdrop object set as an obstacle.</i>
	depth_charge	<i>Depth charge – player's projectile, an active object with three different animations and two movement sets (Pinball movement and Path movement).</i>
	health_counter	<i>Another counter. Acts as our player's healthbar – shows how much mines can our ship take before falling apart.</i>
	high_water	<i>Additional object that will help us control the whole game – it is placed right above the "water_QB" object.</i>
	mine	<i>Unfriendlies' projectile – it is shot from the submarine, contains two movement sets (both are Path movements). It has an Alterable Value ("Timer"), set to 5.</i>
	shadow	<i>Active object that plays the role of the submarine's shadow – it follows the submarine's x position.</i>
	sky_QB	<i>A Quick Backdrop object (set as a vertical gradient), acting as the sky.</i>
	splash	<i>A "water splash" effect – it is displayed when player's depth charges hit the water surface.</i>
	SubHunter3000	<i>Player's ship, set to "Eight directions" movement with only two directions active (left and right). It's speed is set to 30, Deceleration to 20, Acceleration to 18 -</i>
	Submarine	<i>'Tis the enemy, lad! Argh! Sink it or you'll be go to the bottom yourself, I tell 'ya! An active object with two path movements and a single Alterable Value ("Health").</i>
	water_QB	<i>A Quick Backdrop object (set to vertical gradient), acting as the water.</i>

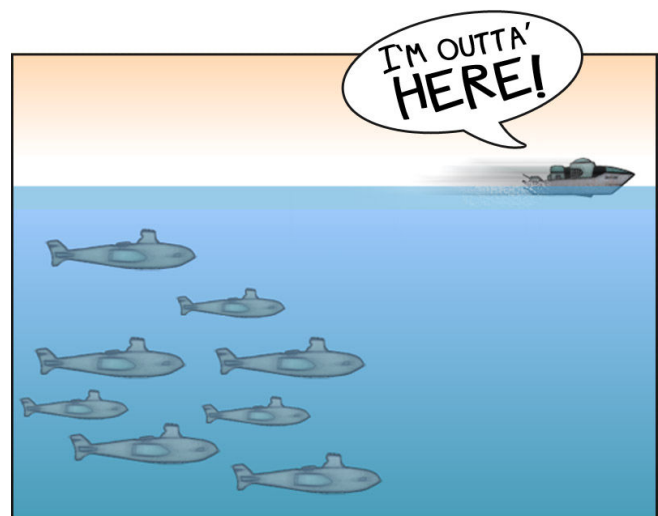
Part III: Taking a break to think it all through

OK, you now know which object does what – and you’re pretty darn ready to code all of this, right? Wrong! Yeah, you would surely manage to script all the events needed, even now, but you would also skip an important part of this tutorial – understanding why all of this works the way it does. Don’t rush to the coding part yet, lad! Take a second to think it all through, to check all the settings on your own, to explore things a bit. You’ve got that little explorer inside of ya’, right? Sure you do – every pirate does! So... Let’s go exploring!

A thing about multiple movements

Aren’t ya’ just a ‘bitty curious why does the *Submarine* object has **TWO movements**, as stated above? Or the *mine* object? Or the *depth_charge* one? Playing with multiple movements isn’t just fun for fun’s sake – using them really widens your range of coding possibilities.

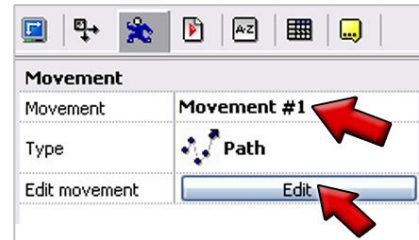
Firstly, let’s take a look at what Multimedia Fusion’s 2 Help has to say about this: “*In Multimedia Fusion 2, an*



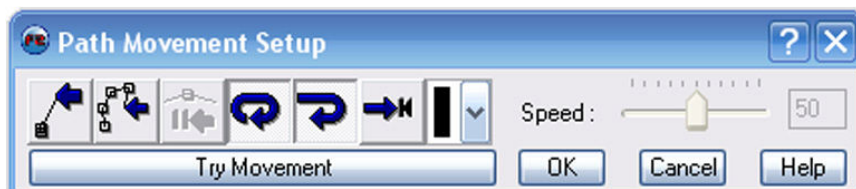
object can have more than one movement. These movements are stored in the object as a list of movements. For example, the first movement can be a path movement, and the second a ball movement. With the actions in the event editor, you can switch from one movement to another”. Pretty simple? Sure. But pretty powerful too. Just think of the things that you can accomplish with this! Player’s movement should change from *Eight Directions* to *Race Car* movement once his spaceship is upgraded? Not a problem! Just set up these two movements and switch between them (via the Event Editor) when player’s character gets near the Upgrade Station and a specific key is pressed.

Now, let’s get back to our frame and analyze the three aforementioned objects – the *Submarine*, *mine* and *depth_charge* ones. Firstly, select the **Submarine** object and go to the **Properties toolbar**. Notice that it has a **Fade Out** animation set up (a smooth Fade to background, set to 2 seconds). Select the **Movement tab** (third from the left, the one with the little man running). This toolbar window consists of three elements: a drop-down list of all movements set within this object, which enables you to both select the specific movement and

add/remove new movements (just click on the *Movement #1* text and an additional button will appear right by the list); another drop-down list – titled *Type* – from which you can select the movement type you want for your object; and an *Edit* button, which enables you to set the preferences for the selected movement. Now, select the first movement (*Movement #1* – it should be selected by default), notice that it's type is set to *Path*. Click the *Edit* button.

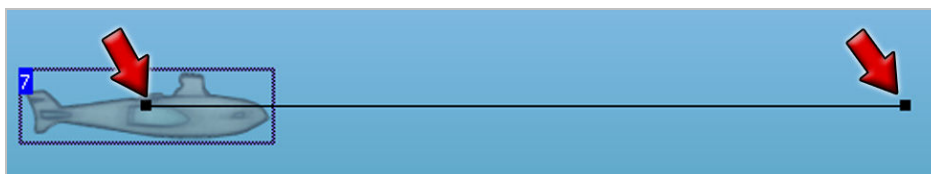


Somewhere in your frame a new window should appear – the **Path Setup** window, which enables you to freely configure your Path movement. It should look like this:



Looks pretty simple, doesn't it? The first two rectangular buttons (counting from left) can be used to add new lines to our movement, either by drawing it with singular clicks (first button), or by taping the mouse movement for as long as the mouse button is pushed (second button). Buttons further to the right can be used to set the preferences for our movement – at this moment it is set to **reverse the object** at the end of the path and **loop** the whole thing.

Now, take a look at our *Submarine* object and notice the black line drawn across the frame, between the two rectangular points:

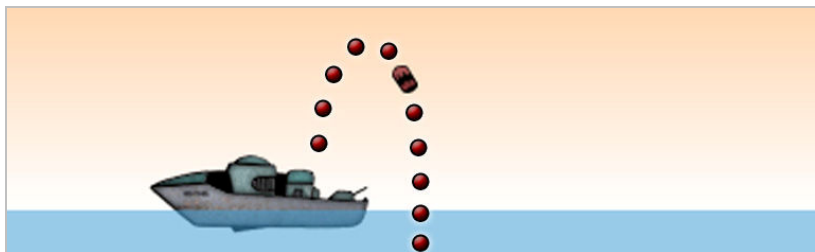


This is the path that our underwater foe will follow. Pretty simple, right? Click *Cancel* to close the *Path Movement Setup* window – there's no more to see here.

Now, let's get back to the **Properties toolbar**. Select the second movement from the list (*Movement #2*) – note that it's yet another Path movement – and click the *Edit* button. Observe how this movement is different from the first one – Submarine's path is vertical this time, there's

more nodes (not just two), with different path speeds between them. As you have most probably guessed by now – this movement will be used once the enemy sub is seriously damaged & sinking, exchanging “hello’s” with the rock-solid sea bottom.

Now, let’s take a look at the *depth_charge* object, which also has two movements set – one of them is a **Pinball movement**, which can be a little confusing at first, but will become obvious in sheer seconds. As MMF2’s Help states: “*The Pinball movement makes the ball move like a pinball: a bouncing ball with gravity*”, which means that it’s a movement that can be used for anything that needs a bit of a gravitational pull. But why would I apply it to a depth charge? Well, the reason is pretty straightforward. I want the charge to be thrown – or catapulted – from the *SubHunter 3000*, not just released into water. In other words – I want it to look like this:



...And the easiest and fastest way to do this is by using the *Pinball movement*. If it was a different multimedia authoring tool you’d most probably had to write at least three lines of code with at least two mathematical equations to achieve the same effect. But with MMF2, all you need are three clicks. Long live MMF2, lads! Argh!

If you wish – you can explore the second movement on your own, as well as the two movements of the *mine* object – but that’s not really necessary, since they use the same pattern as the *Submarine* object does – Path movements that will be initiated when a condition is flagged as *true*. If you’re curious about anything – don’t hesitate to check it out. Take your time – this PDF document isn’t going anywhere, right?

Here’s a quick list of things that you should be aware of before going further:

- The *mine* object has an *Alterable Value* named **Timer**, set to **5**.
- The *Submarine* object has an *Alterable Value* named **Health**, set to **12**.
- Both the *ammo_counter* and *health_counter* are Animation-type counters with both their **initial** and **maximum values** set to **3**.
- The *depth_charge*, *mine*, *Submarine* and *SubHunter3000* objects all have **Fade Out animations** set up – they’re set to either **Fade** or **Zoom**.

Part III: It's time for some coding!

Once you're done exploring – save your project and open the **Event Editor**. Hope that it looks familiar to you – at least a bit. If not – it would be best to get familiar with MMF2's user manual or the "Interface Guide" that I've written some time ago. It isn't really necessary, since all of this is pretty self-explanatory, but knowing what's what always helps a bit.

Koobare's MMF-to-paper coding system

Now, let me introduce you to my MMF2-to-paper event-recording system:

IF (Condition): **[Object for the condition] > Condition group > Condition**
THEN (Action): **[Object for the action] > Action group > Action**

Seems pretty simple, right? All the conditions are marked in red color, while actions are written in 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 we'll have:

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:

< Set the Global Value A to 32 >

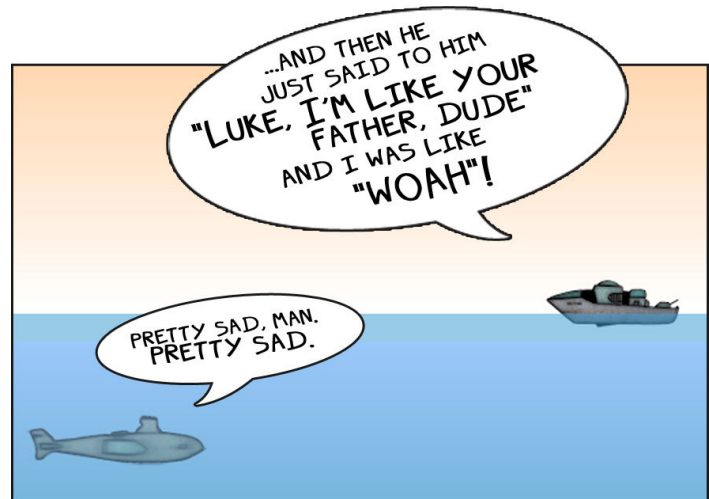
Additional comments, instructions and info will be put in << double angle brackets >>, using a different color:

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

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... Not much philosophy in any of this. And since you already know what to do... Let's start coding!

First things first

1) Firstly, let's start with the traditional "**Start of frame**" event, which will – as soon as the frame starts – play some nice background tune and make sure that our player's health meter (known also as the *health_counter* object) is set correctly. Actually, setting the counters isn't necessary if they already have been set in the *Frame Editor* (and both our counters have – their



initial and maximum values were set to 3, remember?), but I'm always double-checking them, since I usually have giant holes in my memory and keep forgetting about this kind of stuff. Anyway, let's get to it:

IF: [Storyboard Controls] > *Start of frame*

THEN: [health_counter] > *Set Counter*

< input: 3 >

THEN: [Sound Object] > *Music* > *Play music*

<< Select any music from the MMF2's midi library >>

And that's that – our first event is ready. Wasn't that easy? Sure it was. This whole game will be based on pretty simple events – just try to imagine what can be achieved if you'd use more advanced ones!

2) Time for the second event – this one will be based on the **Always** condition. This event will make sure that the sub's shadow (*shadow* object) will always have the same direction and X position as the *Submarine* object.

IF: [Special Object] > *Always*

THEN: [shadow] > *Position* > *Set X position*

< input: X("Submarine") > or click on the *Retrieve data from an object* button, select the [Submarine], choose *Position* > *X Coordinate* from the right-click menu >

THEN: [shadow] > Direction > Select direction

< input: Dir("Submarine") or click on the Retrieve data from an object button,
select the [Submarine], choose Animation > Get direction value from the menu >

As you can see above, you can create Expression Editor's data by either clicking the *Retrieve data from an object* button or by typing in commands from your keyboard. If you're a beginner – I would suggest the "*Retrieve data...*"-button method, but the more time you'll spend in MMF2's Event Editor, the more time-efficient the keyboard method will be. Try to write down all those basic commands on a separate piece of paper and always remember that *dir("object_uno")* will retrieve *object_uno*'s current direction value, while *X("object_uno")* and *Y("object_uno")* will retrieve it's X and Y position.

3) And here's our third event, determining what will happen if the player presses the **space bar** on his keyboard – if it is pressed and the player still has some ammo left (current value of the *ammo_counter* object is greater than 0), a new object (*depth_charge*) is created, a sound is played and 1 is subtracted from the *ammo_counter*:

IF: [Keyboard & Mouse Object] > The Keyboard > Upon pressing a key

<< press SPACE on your keyboard >>

IF: [ammo_counter] > Compare the counter to a value

<< check if it's greater than 0 >>

THEN: [Create New Objects] > Create Object

<< Select the [depth_charge] object >>

<< Set the coordinates to x=-1, y=-4 relatively to the [SubHunter3000] object >>

THEN: [Sound object] > Samples > Play sample

<< Select some kind of a "click" sound from the MMF2's sound library >>

THEN: [ammo_counter] > Subtract from Counter

<< input: 1 >>

4) Remember what was said about player's ammo at the beginning of this tutorial? "*The number of depth charges will be limited, but rechargeable*". Let's script this:

IF: [The Timer Object] > Every

<< Set the timer to 3 seconds >>

IF: [ammo_counter] > Compare the counter to a value

<< check if it's lower than 3 >>

THEN: [ammo_counter] > Add to Counter

<< input: 1 >>

Four done, nineteen to go. If you wish, you can take a look at the screenshot below and compare it with your Event Editor (note that it should look similar, but it doesn't have to be identical – you could have done a few things in a different sequence):

All the events All the objects																			
1	• Start of Frame		✓															✓	
2	• Always										✓								
3	• Upon pressing "Space bar" + <code>key > 0</code>		✓			✓													✓
4	• Every 03"-00 + <code>time < 3</code>																		✓

5) Time to play with the *depth_charge* object a little. This effect will control the graphical side of our *depth_charge* object, and will also change it's movement to the vertical Path movement set previously in the Frame Editor.

IF: *[depth_charge]* > Collisions > *Another object* > *[high_water]*
THEN: *[depth_charge]* > Movement > Multiple movements > *Select movement*
 << select "Movement #2" >>
THEN: *[depth_charge]* > Animation > Change > *Animation Sequence*
 << select "In Water" >>
THEN: *[Create New Objects]* > *Create Object*
 << Select the *[splash]* object >>
 << Set the coordinates to x=-1, y=9 relatively to the *[depth_charge]* object >>
THEN: *[Sound object]* > Samples > *Play sample*
 << Select some kind of a short "splash" sound from the MMF2's sound library >>

6) Here's another event related to the *depth_charge* object, this one determines what happens if our depth charge hits the enemy:

IF: *[depth_charge]* > Collisions > *Another object* > *[Submarine]*
THEN: *[depth_charge]* > Animation > Change > *Animation Sequence*
 << select "Disappearing" >>
THEN: *[Submarine]* > Alterable Values > *Subtract from*
 << subtract 1 from *Health* >>
THEN: *[Sound object]* > Samples > *Play sample*
 << Select some kind of an explosion or hit sound from the MMF2's sound library >>

7) Another "on collision" event for our player's depth charge, this one is played out once there is a collision between the *depth_charge* and a backdrop set to obstacle (like our sea bottom):

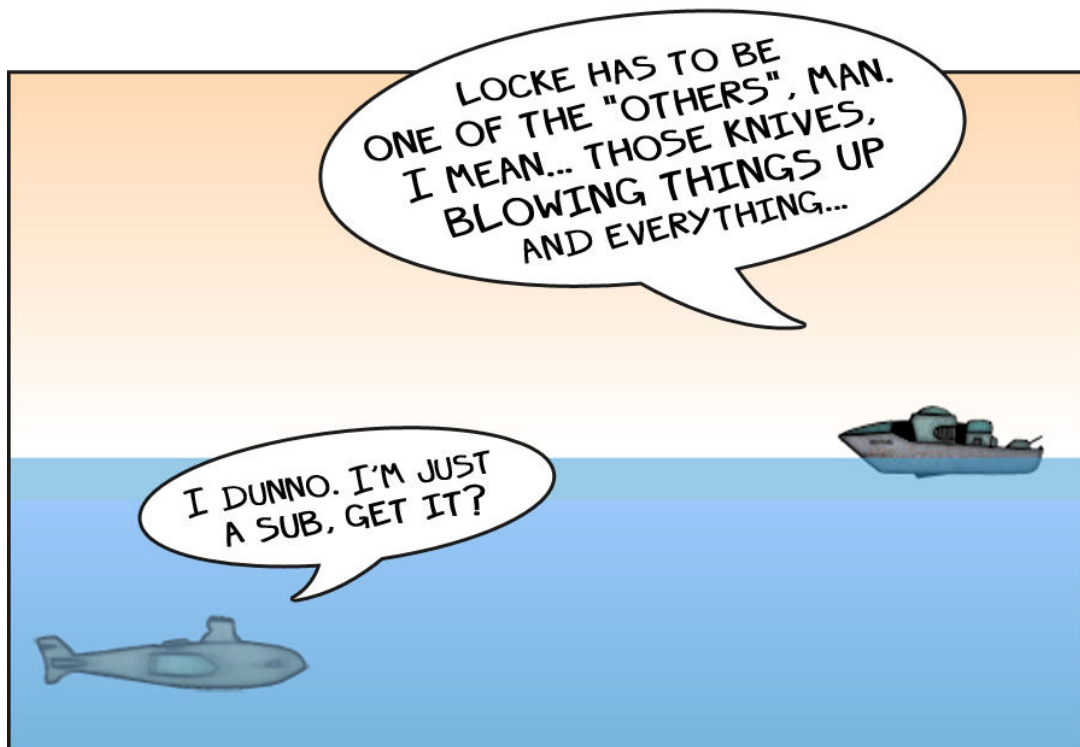
IF: [depth_charge] > Collisions > *Backdrop*

THEN: [depth_charge] > Animation > Change > *Animation Sequence*

<< select "Disappearing" >>

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

<< Select some kind of an explosion or hit sound from the MMF2's sound library >>



8) Yet another *depth_charge* event – this one simply destroys the given object when it's "Disappearing" animation is over.

IF: [depth_charge] > Animation > *Has an animation finished?*

<< select "Disappearing" >>

THEN: [depth_charge] > *Destroy*

9) OK, we've got all the events needed to deploy our depth charges and try to hit the enemy... And here's the enemy shooting back!

IF: [The Timer Object] > *Every*

<< Set the timer to 3.50 seconds >>

IF: [Submarine] > Alterable Values > *Compare to one of the alterable values*

<< check if *Health* is greater than 0 >>

THEN: [Create New Objects] > Create Object
 << Select the [mine] object >>
 << Set the coordinates to x=3, y=-6 relatively to the [Submarine] object >>
THEN: [Sound object] > Samples > Play sample
 << Select some kind of a “click” sound from the MMF2’s sound library >>

10) Here’s a whole bunch of events controlling the collisions of the *mine* object and their outcome (create them one by one):

a)

IF: [mine] > Collisions > Another object > [SubHunter3000]
THEN: [Create New Objects] > Create Object
 << Select the [boom] object >>
 << Set the coordinates to x=0, y=0 relatively to the [mine] object >>
THEN: [mine] > Destroy
THEN: [Sound object] > Samples > Play sample
 << Select some kind of an explosion or hit sound from the MMF2’s sound library >>
THEN: [health_counter] > Subtract from Counter
 << input: 1 >>

b)

IF: [mine] > Collisions > Another object > [depth_charge]
THEN: [depth_charge] > Destroy
THEN: [mine] > Destroy
THEN: [Create New Objects] > Create Object
 << Select the [boom] object >>
 << Set the coordinates to x=0, y=0 relatively to the [mine] object >>
THEN: [Sound object] > Samples > Play sample
 << Select some kind of an explosion or hit sound from the MMF2’s sound library >>

c)

IF: [mine] > Collisions > Another object > [high water]
THEN: [mine] > Movement > Multiple movements > Select movement
 << select “Movement #2” >>
THEN: [Sound object] > Samples > Play sample
 << Select a quiet “splash” or “click” sound from the MMF2’s sound library >>

d)

IF: [mine] > Collisions > Another object > [boom]
THEN: [mine] > Destroy
THEN: [Sound object] > Samples > Play sample
 << Select some kind of an explosion or hit sound from the MMF2’s sound library >>

11) These two events are pretty self-explanatory. Once the animation is over – destroy the selected object. Create them – as usual – one after another.

a)

IF: [boom] > Animation > *Has an animation finished?*

<< select “Stopped” >>

THEN: [boom] > Destroy

b)

IF: [splash] > Animation > *Has an animation finished?*

<< select “Stopped” >>

THEN: [splash] > Destroy

12) Water mines should blow up after a few seconds – that’s what their *Timer* Alterable Value was set up for. This whole thing is accomplished by these two events:

a)

IF: [The Timer Object] > *Every*

<< Set the timer to 1 second >>

THEN: [mine] > Alterable Values > *Subtract from*

<< subtract 1 from *Timer* >>

b)

IF: [mie] > Alterable Values > *Compare to one of the alterable values*

<< check if *Timer* is equal 0 >>

IF: [Special Object] > Limit conditions > *Only one action when event loops*

THEN: [mine] > Destroy

THEN: [Create New Objects] > *Create Object*

<< Select the [boom] object >>

<< Set the coordinates to x=0, y=0 relatively to the [mine] object >>

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

<< Select some kind of an explosion or hit sound from the MMF2’s sound library >>

13) Moving on to the next event... Ever wondered what happens to a submarine when it receives a bit too many depth charges to endure? Yep, it blows up! Ay, lad, ‘tis that fancy brain of yours workin’! But I tell ya’: it not only blows up, but it sinks too! Here, see for yourself:

a)

IF: [Submarine] > Alterable Values > *Compare to one of the alterable values*

<< check if *Health* is equal 0 >>

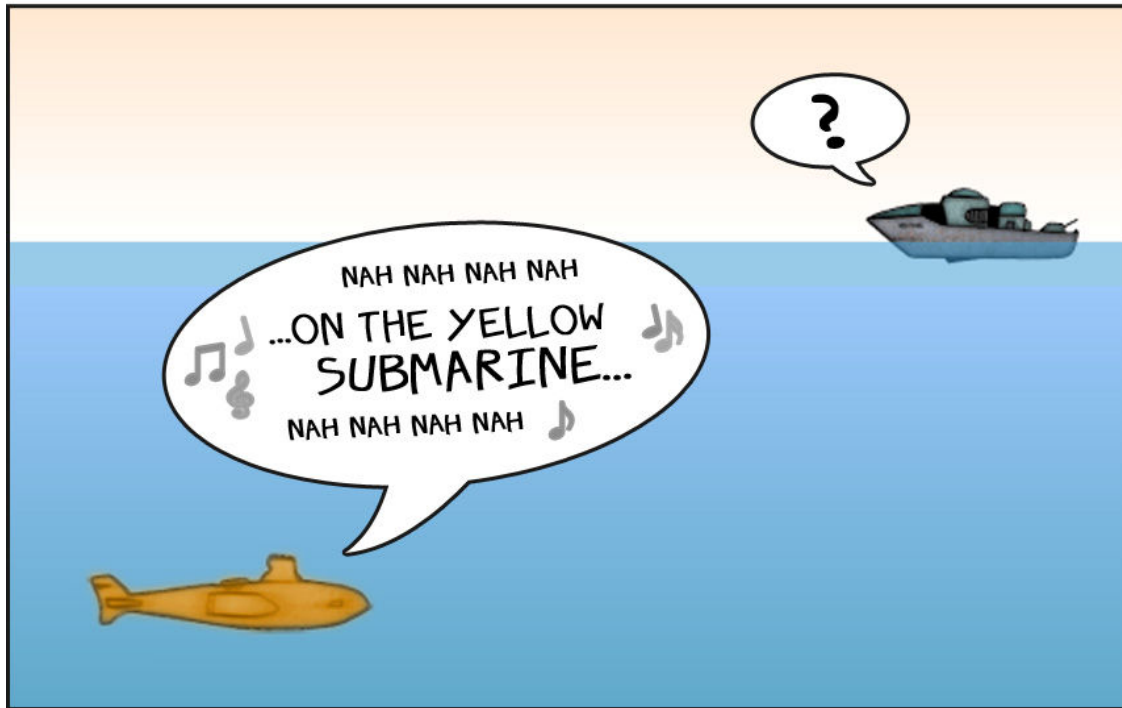
IF: [Special Object] > Limit conditions > *Only one action when event loops*
 THEN: [Submarine] > Movement > Multiple movements > *Select movement*
 << select "Movement #2" >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=46, y=0 relatively to the [Submarine] object >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=-28, y=3 relatively to the [Submarine] object >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=14, y=-8 relatively to the [Submarine] object >>
 THEN: [Sound object] > Samples > *Play sample*
 << Select a big explosion sound from the MMF2's sound library >>

b)

IF: [Submarine] > Collisions > *Another object* > [shadow]
 IF: [Special Object] > Limit conditions > *Only one action when event loops*
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=38, y=6 relatively to the [Submarine] object >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=-42, y=4 relatively to the [Submarine] object >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [boom] object >>
 << Set the coordinates to x=32, y=11 relatively to the [Submarine] object >>
 THEN: [Submarine] > *Destroy*
 THEN: [Sound object] > Samples > *Play sample*
 << Select a big explosion sound from the MMF2's sound library >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [splash] object >>
 << Set the coordinates to x=-32, y=5 relatively to the [Submarine] object >>
 THEN: [Create New Objects] > *Create Object*
 << Select the [splash] object >>
 << Set the coordinates to x=55, y=3 relatively to the [Submarine] object >>

c)

IF: [Submarine] > Pick or count > *Have all "Submarine" objects been destroyed*
 THEN: [Storyboard Controls] > *Next frame*



14) Better save your project, since... We're almost at the end! Here you go – last three events to code and you can play your very own Risky Waters game!

a)

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

<< check if it's equal 0 >>

IF: [Special Object] > *Limit conditions* > *Only one action when event loops*

THEN: [Create New Objects] > *Create Object*

<< Select the [boom] object >>

<< Set the coordinates to x=-39, y=3 relatively to the [SubHunter3000] object >>

THEN: [Create New Objects] > *Create Object*

<< Select the [boom] object >>

<< Set the coordinates to x=27, y=-7 relatively to the [SubHunter3000] object >>

THEN: [Create New Objects] > *Create Object*

<< Select the [boom] object >>

<< Set the coordinates to x=-14, y=-18 relatively to the [SubHunter3000] object >>

THEN: [Create New Objects] > *Create Object*

<< Select the [boom] object >>

<< Set the coordinates to x=2, y=16 relatively to the [SubHunter3000] object >>

THEN: [SubHunter3000] > *Destroy*

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

<< Select a big explosion sound from the MMF2's sound library >>

THEN: [Create New Objects] > Create Object

<< Select the [splash] object >>

<< Set the coordinates to x=-46, y=13 relatively to the [SubHunter3000] object >>

THEN: [Create New Objects] > Create Object

<< Select the [splash] object >>

<< Set the coordinates to x=14, y=11 relatively to the [SubHunter3000] object >>

b)

IF: [SubHunter3000] > Pick or count > *Have all "SubHunter3000" objects been destroyed*

THEN: [Storyboard Controls] > Restart the current frame

c)

IF: [SubHunter3000] > Position > *Test position of "SubHunter3000"*

<< 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: [SubHunter3000] > Movement > Bounce

And that's all, folks! Yep, we're at the finish line! The End! Finito! Fin! Koniec! Now you have your own Risky Waters game to play with, to enhance and extend! Save it to your hard drive, build some new levels, add more enemies, change timers and counter values! And always remember: practice makes perfect!

Thanks for your time and see you again soon!

Cheers!

Koobare

marchewkow@gmail.com

Sounds by the Freesound Project

If you have any questions, suggestions or just need help –

mail me at marchewkow@gmail.com

You have been reading...

RISKY WATERS

a Multimedia Fusion 2 tutorial



BROUGHT TO YOU BY

KOOBARE

CLICKTEAM'S funkiest

~~mercenary~~

pirate

Created for Multimedia Fusion 2 & Multimedia Fusion 2: Developer

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