

Q3A Electronics Manual

A REFERENCE OF THOUGHTS

By David 'skOre' Deutsch

Table of Contents

Q3A Electronics Manual

Table of Contents

INTRODUCTION

1 - MECHANICS

1.1 - THE BASICS

COMPLEX FUNC_ING, TRIGGER_ING AND TARGET_ING
HOW TO... CREATE POP UP EFFECTS

1.2 - ADVANCED TRIGGERING

1.3 - MORE COMPLEX EXAMPLES OF AT

THE TELEPORT-TUNNEL

FURTHERMORE: THE TELEPORT-TUNNEL

INTERLUDE: FIRST STEPS DONE

2 - BASIC ELECTRONICS

2.1 - Q3A ELECTRONICS

THE AND-GATE

THE NOT-GATE

THE OR-GATE AND INTERLUDE

THE EQUIV-GATE

THE XOR-GATE

2.2 – GATES AND BEYOND

FLIP FLOPS

2.X - THE USE OF Q3A ELECTRONICS

INTRODUCTION

As a mapper, I have always been interested in exploring new things, especially with respect to the limits of what is possible. One thing that has always bothered me about the vanilla Quake 3 engine is the fact that its flexibility in regards to scripting events are, quite frankly, pretty limited. You can have events trigger other events, but that's the extent of the scripting in Q3. Ever since I started mapping, I've always been attracted to this single idea, so I finally decided it was time to find out what was possible.

So I decided I would wishfully gaze over to the electronic world, where everything seems to be possible--triggers, gates, networking and arraying. But then I realized, *why shouldn't* it be possible in Q3A? We have physics in Q3A, don't we? Realizing this point, I then attempted to make it all possible, having only the entity maximum as the limit. I hope you enjoy this introduction to the world of Q3A Electronics and discover how to be inventive with scripting yourself!

It is essential that you check out the links under 'Must Reads' (Link Section) as a knowledge of the material contained in these is required before you can move on to the heavier stuff.

Another thing I should mention before you read this is that you shouldn't assume that this is a complete guide or something that is 100% right. I am always open to and happy about suggestions in order to make this a better guide.

1 – MECHANICS

1.1 - THE BASICS

There is a whole world to explore, but I will try start out with the basics, that I call mechanics, and make it more complex as we move along. The first step is to push what you think is possible with the basic things given to you.

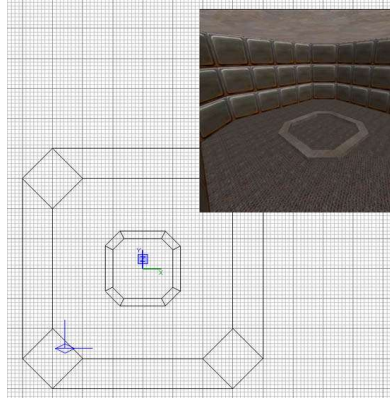
COMPLEX FUNC_ING, TRIGGER_ING AND TARGET_ING

Starting off alphabetically, the first thing are **func_doors**, which play a big part in all what I'm talking about later on. The main problem with **func_doors** is that they have that hard-coded problem of making a sound if used. Some time ago I tried to deal with this problem and came across *4days* tutorial about alternative door-sounds (check the Thanks-Section), which made me write a more complex and detailed tutorial about pop-up effects:

HOW TO... CREATE POP UP EFFECTS

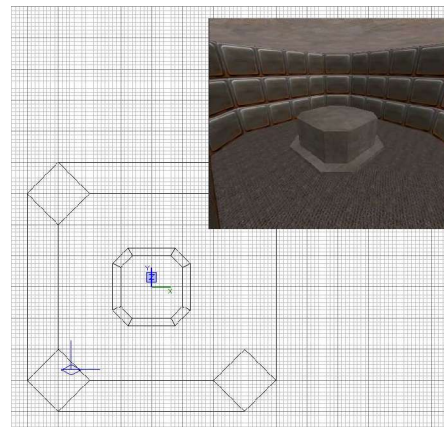
You can make everything just pop up in your level (except things that are already entities, such as rotating or bobbing brushes and so on) by just using silent func_doors. The thing with 'silent func_doors-popups' is that you have to do some maths to get the wished effect.

Let's start with something simple, a brush popping up in a simple room



The test room

What we want to achieve, is that a brush pops up in the middle of the room, when somebody enters it.



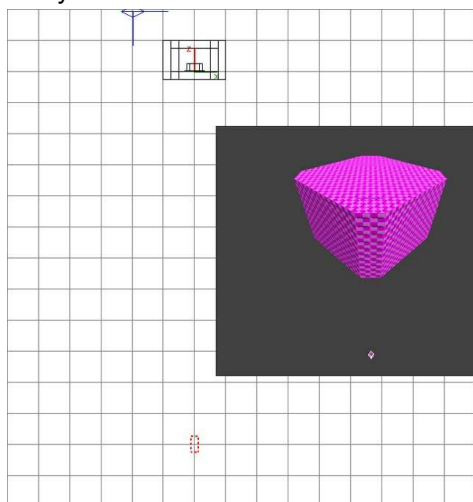
The test room with the pop-up brush

So we start off placing the brush

The real maths starts here, but let's do a short cut to the what's-it-all-about. What we use by doing this is that the door-sound is played exactly in the middle of the brush. We combine that with the fact that you can combine brushes that are not touching each other to one entity.

So now place a brush widely outside the room, it must be very wide, maybe about 3000 units.

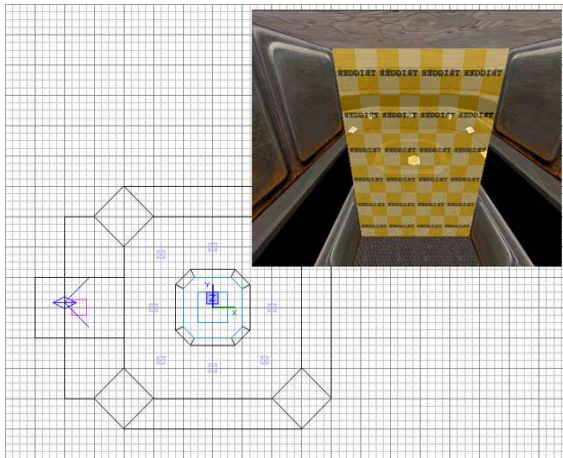
Select both brushes and convert them to one func_door.



And now to the part with the maths (very light algebra) first we have to know how tall this func_door is, so we select it and press 'Q'

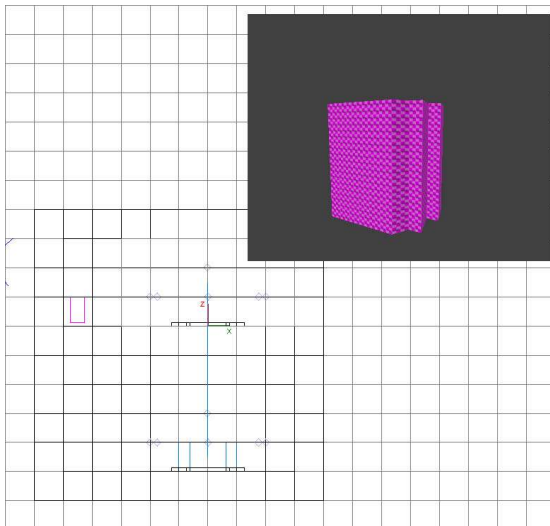


We have to know this, because the func_door needs a 'lip' keyword, otherwise it would move 3200 (yes, 3200 - 8 by default) units higher... we want to move it out of the room, in this case exactly 320 (note that number too)



The final room with the trigger

Then do the hard algebra $3200 - 320 = 2880$ which will be our 'lip' – amount. The other keys are: speed = -1 (immediate pop up) angle = -1 (has to pop 'up') wait = 1 (All values below 1 are working bogus). Then we create a playerspawn and the room trigger->a trigger_multiple and connect the entities, by first selecting the trigger, then the door and then pressing ctrl+k The last problem now, is that the appearing brush will be completely black, as it has no light outside the map.



The two rooms

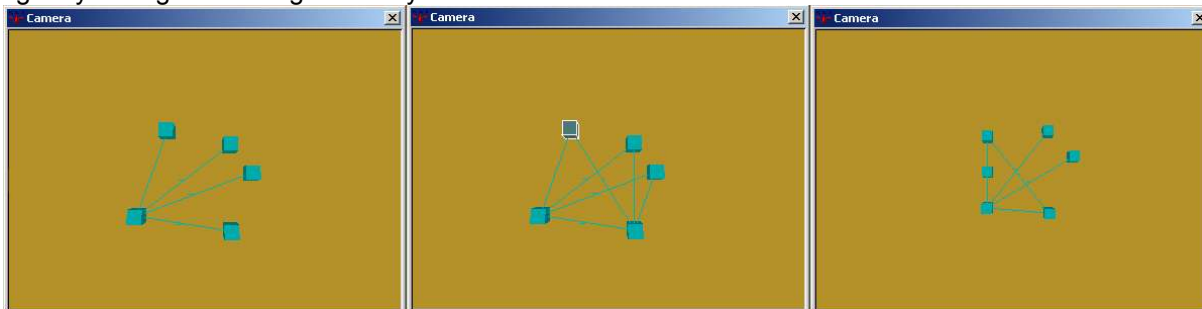
You could go around this problem by putting the func_door into the map, and activate the start_open key and so on, but than you would produce shadows on the floor where it should spawn, so we just create a similar lightmap-simulation-room below the original one and our problem is solved.

The document package includes the zipped .map- and .pk3-files.

1.2 - ADVANCED TRIGGERING

Now its time to dig these other func_s, trigger_s and target_s to be ready for the real electronics, this might be a long read, but could clear many things to be more confident while dealing with triggers.

A big thing is the **linking of triggers and targets**. The Radiant is very clever in dealing with huge and disturbing link-fests of triggers and targets, but has got certain limitations. For linking triggers, targets and funcs into networks, you usually use target_relays, this example shows in what trouble you can get by linking them too generously:

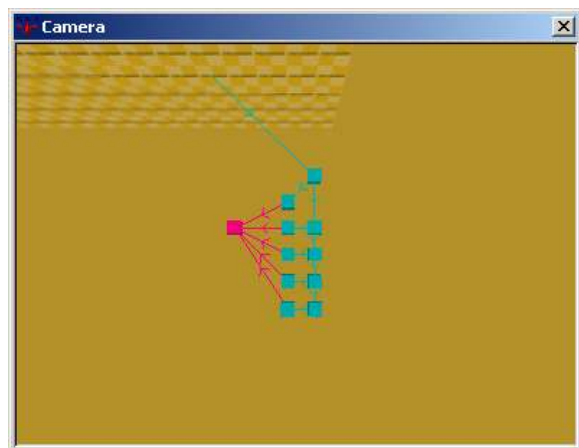


In the second picture, I have just linked the bottom right relay with the top left one, and it all started to get messy. The whole thing is strictly logical, the bottom left relay is the target t2 and all the other relays thus get targetname t2, but the only possible way for the interaction shown in the second picture is that the bottom right relay also becomes target t2, making it an aim for all the other relays too.

The solution is stated in picture 3, in which you can see how only one relay for separation can clear things up a bit. So always check what you are doing and if it is causing any problems.

Something that I found interesting was that the **trigger_always** has somehow undergone wrong naming. Its not a trigger that continuously fires its targets, but indeed a trigger_once (!) that fires its targets once the map has started.

Weapon_s and Item_s and so on can be spawned by being triggered, two things that you have to consider are, that it will *start* spawning if triggered, so you will have to set 'wait -1' to have it *only* spawned if triggered. The second thing is that the weapon will spawn and spawn again if triggered, refer to section [FIX ME] for a more detailed solution.



The use of **shooter_s** is a bit tricky, for example if you want to have a trigger_multiple unleash salvos of plasma on the player tapping into it, it will occur to you that these salvos come each half second, as this seems to be the rate the engine questions the triggers whether or not they are triggered and what it has to do. Bypassing this is easy, have your trigger_multiple trigger a target_relay (this is always a good start), passing the signal on to a relay to the shooter and to a target_delay with a 'wait'-key set to 0.1 . This target delay now triggers the shooter via a relay and another target_delay with the same values and so on to the fourth target_delay. And now you really have a shooter that can be called like this.

The document package includes shooters that fire at various rates (shooter_5in1 being a circuit that fires a relay 5 times in one second)

Another good thing if it comes to teamplay are **target_relays**, as these can be switched to be activated by one team color. If you put one of these between a teleporter and its target, you can make it a one-team-teleporter

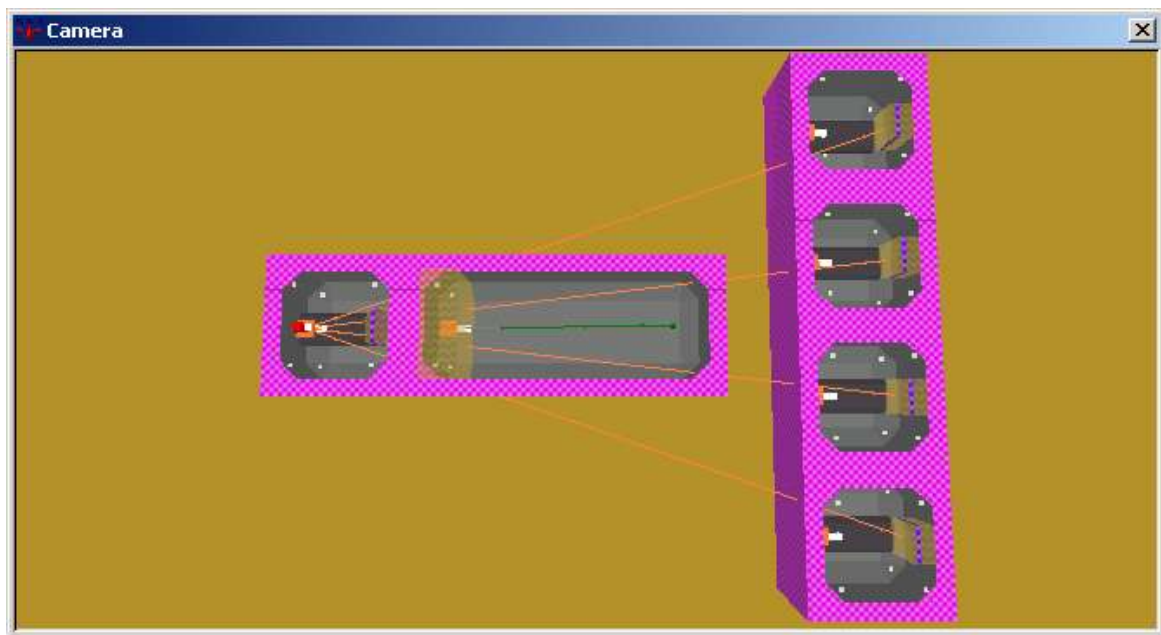
1.3 - MORE COMPLEX EXAMPLES OF AT

Now coming to more interesting things – what is possible with what we got by now, just some examples off my mind.

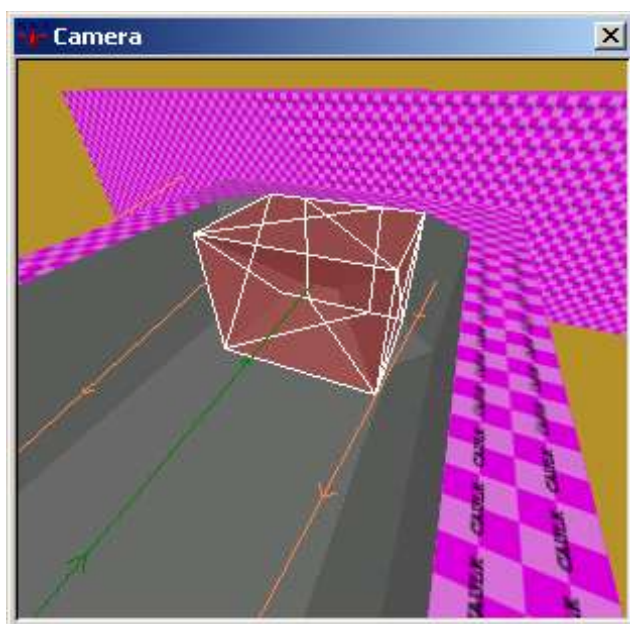
The Teleport-Tunnel

This will be a teleport tunnel with random aim, so that the player will end up in one out of four possible **misc_teleporter_dest**'s. But what's the big deal – just link the **trigger_teleport** to the four aims and you will have it random.

But I want to achieve something else, I want the player to be able to follow his enemy through that tunnel to a certain extend. So now we got a challenge...



We start off with this tunnel, one room on the left, leading into it and four rooms leading into the left room. Also we got some lights and a pusher around the **misc_teleporter_dest** in the tunnel.



At the end of this tunnel, we place four brushes with a 45 degree inclined side, facing up, down, left and right. If a player gets fired against one of these, he will directly be facing a different angle, so that now we place a **trigger_teleport** at every of the four sides, link them to the four **rooms** and convert the four brushes to **func_doors** as we want to trigger them later on. You have to give these keys to the **func_doors**:

Angle = -1 / -2 / 90 / 270 (depending on which brush you take)

Wait = 5 (we want it to change every 5secs)

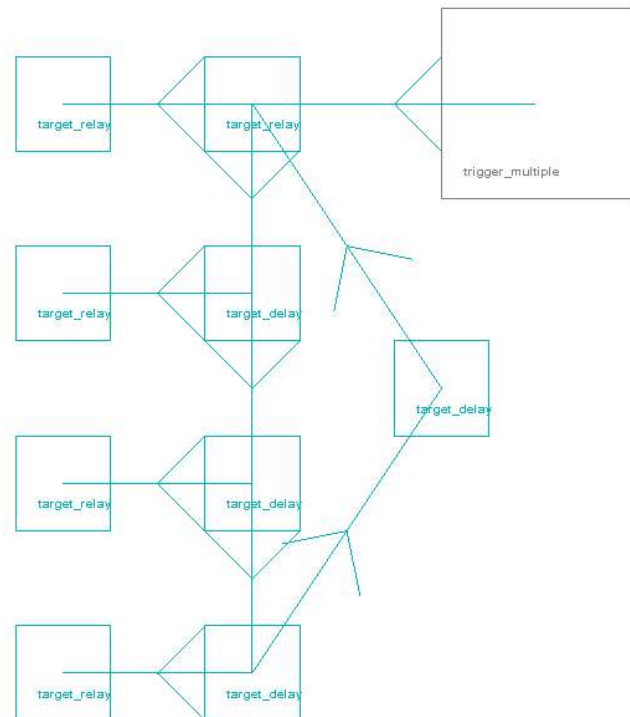
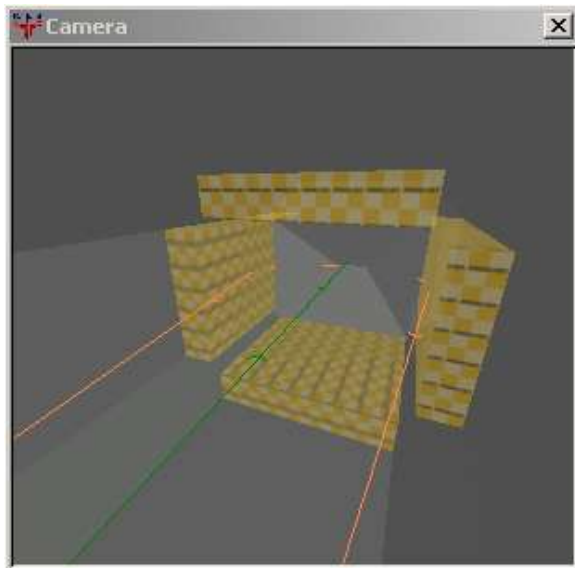
Speed = -1 (immediately)

Spawnflags = 1 (start_open)

Lip = 0

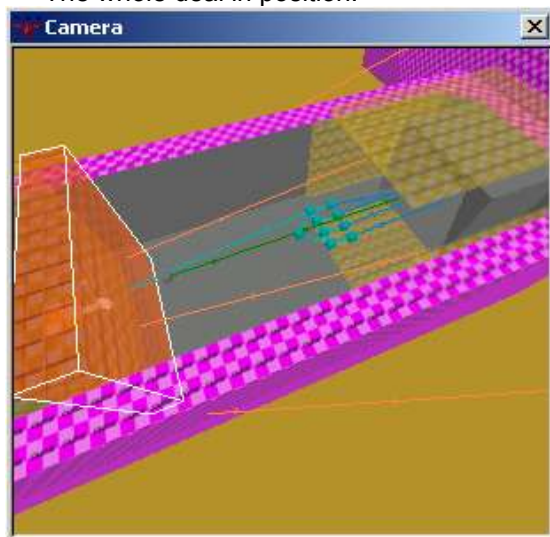
The last two things will make them start outside of the tunnel and let them go in if triggered.

What we want to achieve with the following targeting is, that the whole system changes the direction every five seconds.



The left relays will give the signal on to the func_doors, the trigger_multiple has a 'wait'-key set to -1 and will start the mechanism as soon as players want to use it. This signal will then activate the first func_door and be passed on to the first target_delay, waiting five seconds and then triggering its door and the next target_delay and so on. In the end, a final target_delay will loop the event again and again.

The whole deal in position:



The last thing is to decorate the four rooms so that they are distinguishable, I chose coloured lights.

Furthermore: The Teleport-Tunnel

There are many more possibilities left, you could add more teleporter-aims to it by building what we've build at the end of the tunnel again before it and then adding this to the target-chain. Just keep in mind that a longer the tunnel will produce more air control thus producing a higher chaos-factor.

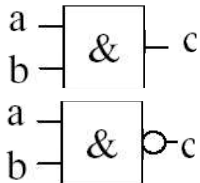
Another thing you could do with a different tunnel, is a prevention of getting chased through the tunnel. This can be made by having a trigger_multiple triggering a **func_door** that will guide the player behind the one that activated the **trigger_multiple** to a different destination. I will add more complex ideas to the tunnel-idea later when the guide gets more complex.

Interlude – First steps done

So now a short stop for taking a breath, I try to make the manual go harder step by step and you've just gotten through the first steps. Now it might start to get abstract for you. It might be very helpful to have knowledge about electronics, but it is not necessary (hey, maybe you can learn them through this...).

2 – BASIC ELECTRONICS

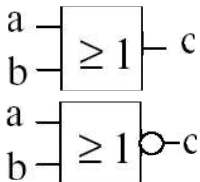
In real electronics, circuits are build by gates, which I will explain here very shortly. They are always about what signal will one gate put out if triggered in a certain way. This is strictly binary system from here on. To clarify the thing, I added charts to understand, in what cases the gates react how, in detail.



x	y	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

AND and NAND-Gate

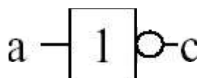
If all incoming signals are = 1 the outgoing signal is = 1. The NAND-Gate is the other way round.



x	y	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

OR and NOR-Gate

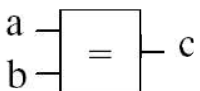
If one or more incoming signals are = 1 the outgoing signal is = 1. The NOR-Gate is the other way round.



x	NOT
0	1
1	0

NOT-Gate

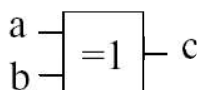
The outgoing signal is the opposite of the incoming signal.



x	y	EQUIV
0	0	1
0	1	0
1	0	0
1	1	1

Equivalence-Gate

If the incoming signals are the same (all 1 or 0), the outgoing signal is = 1.



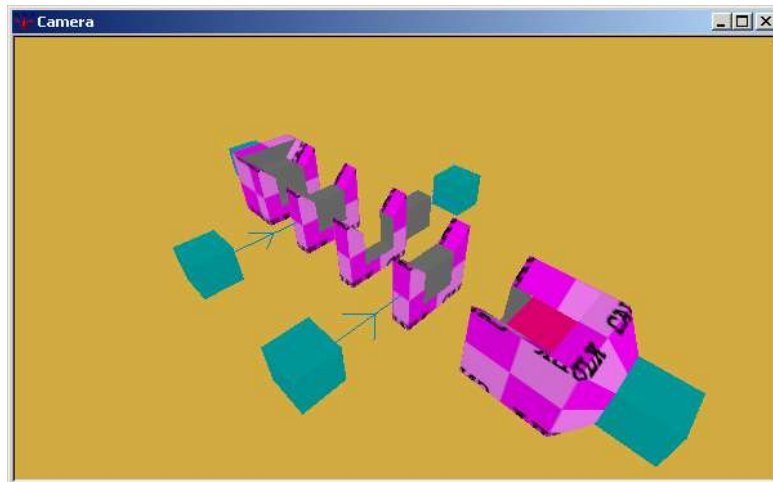
x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR-Gate

If one incoming signal is = 1 the outgoing signal is = 1.

2.1 – Q3A ELECTRONICS

From now on it's just an easy step. The interesting thing is, that what we will do now is exactly what you would do in reality, we are building these gates out of brushes:



The AND-Gate

This is basically the idea I wrote the whole manual for, this is the most important thing to learn. What we got (from right to left) is, all triggered by a **target_relay** for separation:

- a **shooter_plasma**
- one input signal, represented by a **func_door**
- one blocking signal, also represented by a **func_door**
- another input signal
- the **func_button** that will be triggered

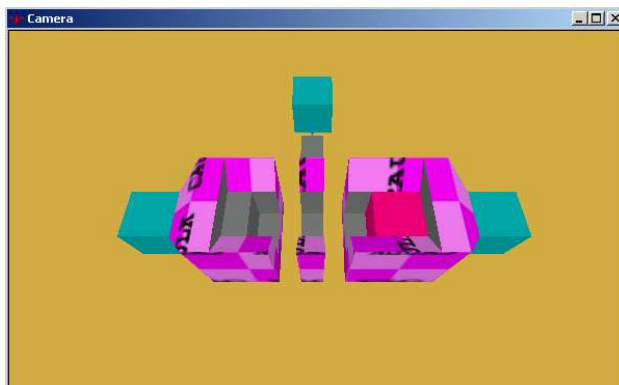
Note that these are just like 'bricks' to be built together.

Maybe you already get the point, the **func_button** will be triggered if the doors are out of its way, so an event will be triggered if other events happened. If you want a certain thing to not let this all happen, you can use a blocker. Some things that can be tweaked on this:

- The quickness of the gate – settable by the rate that the **shooter_plasma** fires (refer to chapter 1.2 for further information)
- The time in which the events must be done – settable by the 'wait'-key of the **func_doors**
- The rate of how fast this can be triggered – settable by the wait key of the **func_button**

The NOT-Gate

This one is needed for that N* thing, it's not very complex. What we need is just this:



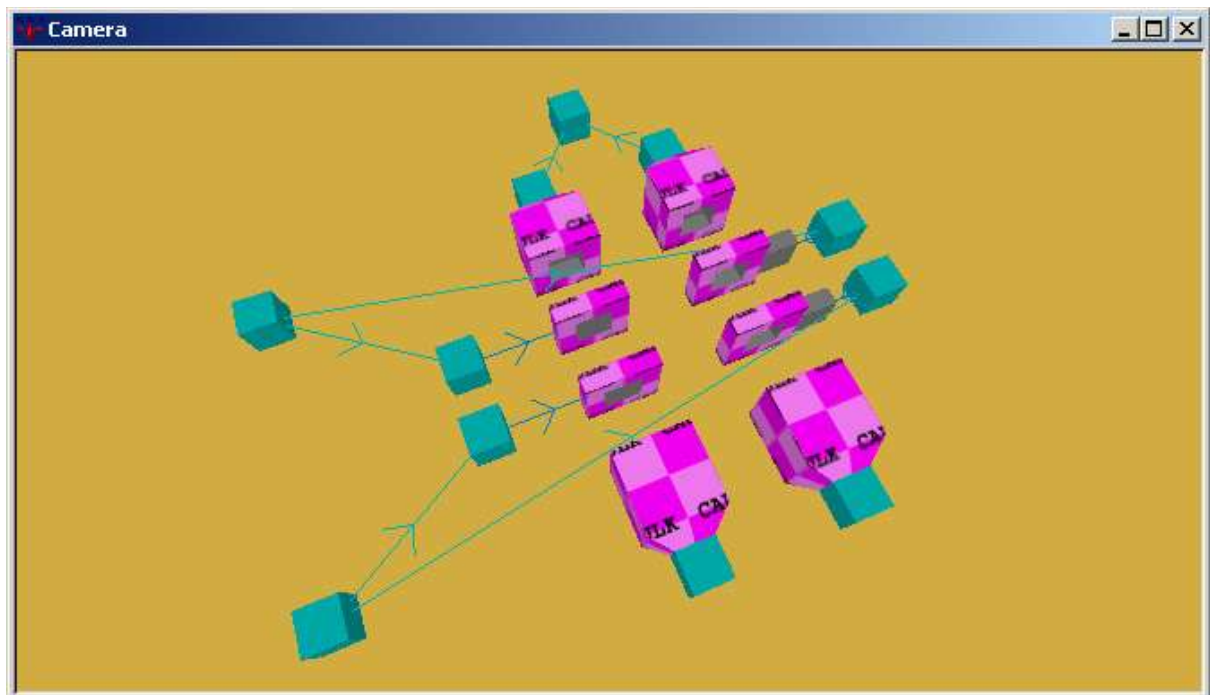
Ha! What a surprise! It looks just like the thing before! Nah, it looks like it, but it's used different. Instead of linking the inputs to the door-parts, you link the signal of the previous gate to it. Now the button turns into the new outgoing signal. So now if there is no signal, the button will get activated (always remember that there needs to be a constant signal for this thing to stay active [the thing on the right])

The OR-Gate and Interlude

The OR-Gate is not hard to do at all, as stated before, just link the inputs to one output. We now have got the first five gates, no big deal, but the last two are another challenge.

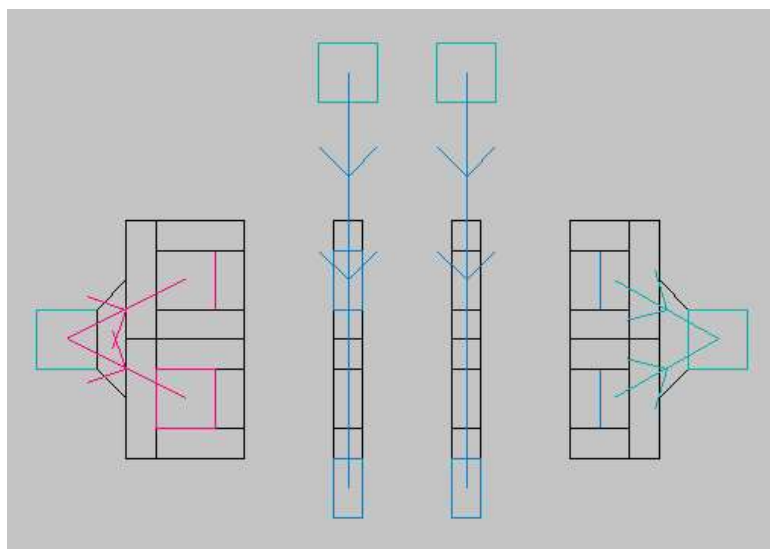
The Equivalence-Gate

We'll do the equivalence gate for quick, so that your brain is ready for the hardcore XOR stuff.



One block on the left, triggered if both signals are 1, one block on the right, triggered if both signals are 0. Also, it won't work if just one is 1 or 0, how about that?

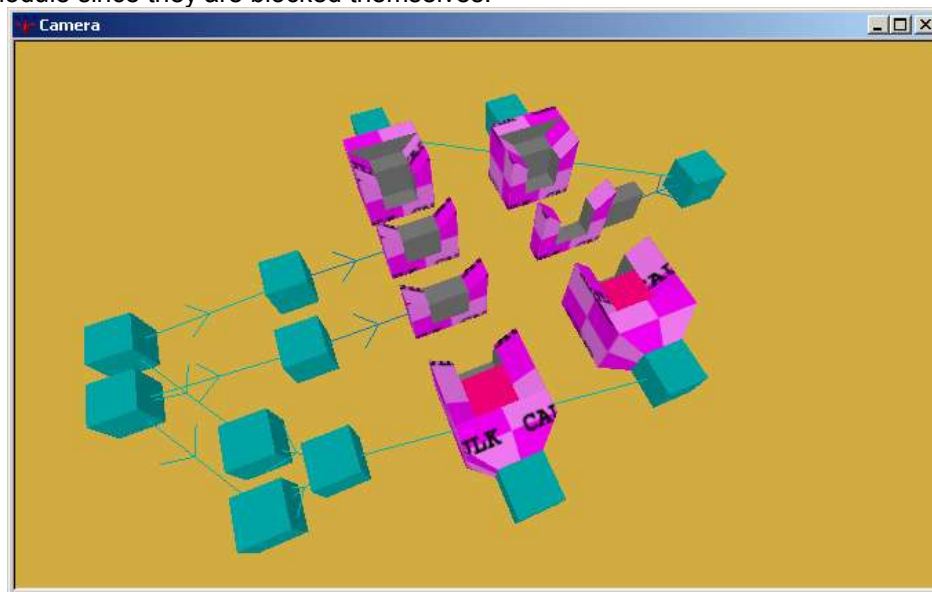
This is the entity-reduced version (document_pages ++):



The XOR-Gate

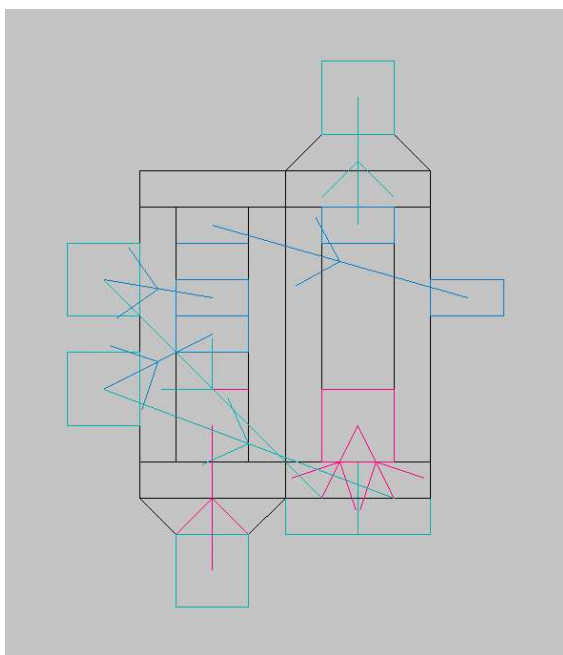
We now come to the most complex trigger (well, most complex for me to find out...) but in the end it is one of the most helpful ones.

I first thought that it could be easy done by doing one module for each input made of an AND-Gate with a NOT-possibility in it. The output of this module would trigger any other NOT-possibility of the other modules. But this version makes no sense as there would be no possibility for other modules to stop a first module since they are blocked themselves.



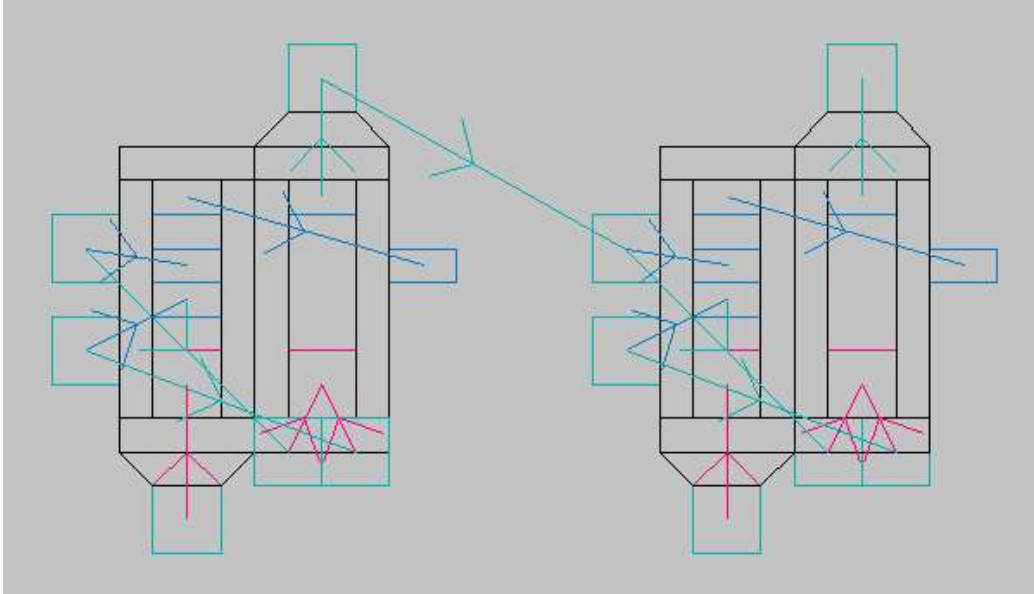
So this is the entity-expensive variation of an XOR-Gate, just to make the point easier to get. We've got 11 relays to guide the signals, two shooters, two buttons and 3 doors. The left block with the two doors is used to detect if both signals are active, which would mean that no XOR is given. The block of three relays on the bottom left of the picture is a genuine OR-Gate triggering the right block. If there is a signal, the shooter will hit the button; if the left block gets activated it will be blocked, simple as that.

This way to do the XOR is also a good since you can now just take this single element and use it for more complex XOR-Gates. But first, the clean and reduced version:

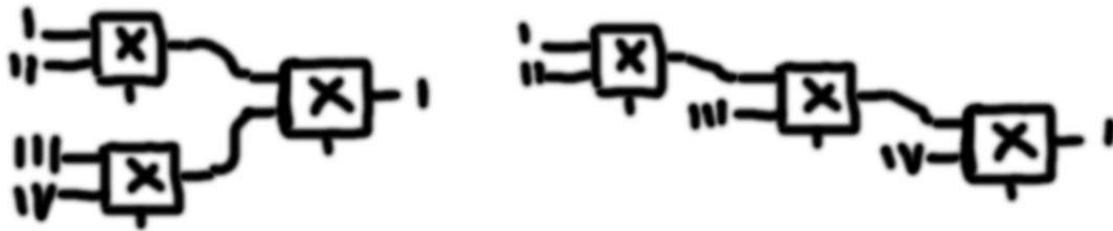


On the left the two inputs, on the bottom the activation input, and at the top the output. This version is also reduced from 18 entities to just 13. This is also the only gate coming in a module format as it is not expandable as the other ones (just copying the doors-modules).

To build more than two inputs is now pretty easy, the XOR-Gate is like a module that can be linked forever:



Just link the output into one input of the next XOR-Gate. To even get more inputs, you can use one of these strategies, depending on your taste:



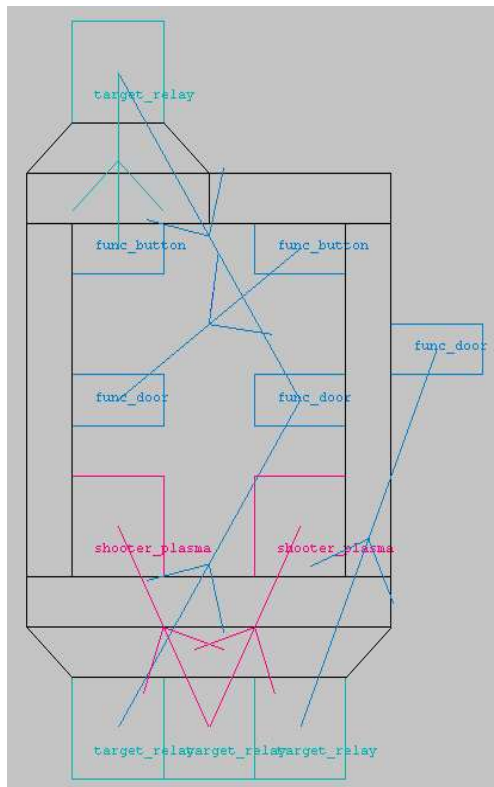
Keep in Mind:

These triggers won't work if they are somewhere in the void. You would be good off building a trigger-room anyways.

2.2 – GATES AND BEYOND

Flip Flops

Flip flops were a new challenge to me, being still enthusiastic about the gates. After the first gates, I thought of how nice it would be to have a storage for values. And as building the gates worked so well, I tried my best on flip flops, being the answer to this question in the electronic world.

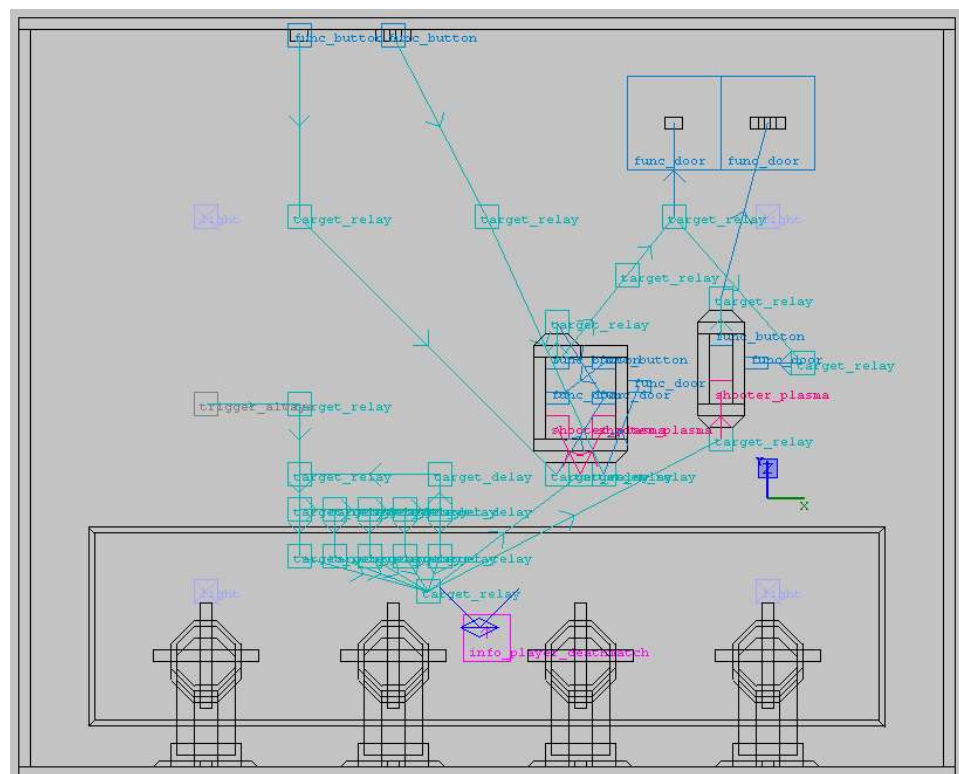


This is already the entity-condensed version of my flip flop layout. It is basically a Set/Reset flip flop. It has two inputs, set (the bottom left relay), reset (the bottom right relay) and one output (the upper relay). The relay in the middle is activating the shooters, so this is setting the quickness of the flip flops reaction (in combination with how quick the doors react and how long they wait off course).

Storing a value is achieved by one gate triggering itself, which can be stopped by the reset relay.

Note, that this is not a flip flop that can “be asked to return whether it stores a '0' or a '1'”, it will continuously provide its state to your circuits.

The document package holds this gate and a sample map to show how it can be used (see the picture below)

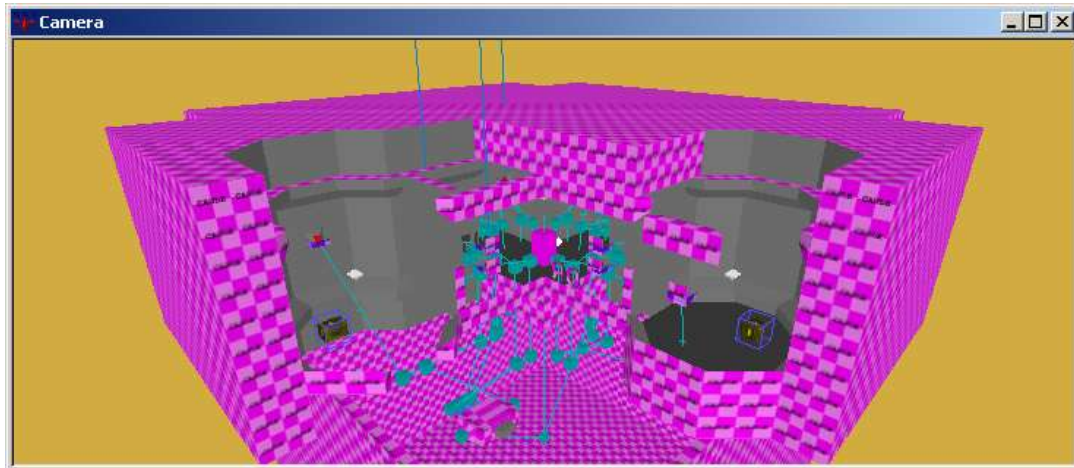


2.X – THE USE OF Q3A ELECTRONICS

“Good stuff, but what can I do with it ?” That's an interesting question. The simple answer is “ just about anything.” For your pleasure, I have thought of some things:

Different types of doors

This demo map shows some doors that are opened by 4 buttons, each with a different type of linking. This demonstrates AND-, OR-, and EQUAL-Gates. You can add a fourth one if you want. I also kind of scripted the doors, so that they open in 3 steps.



The End of use ?

At this point, this guide will stop. Mostly because I seem to be unable to kick myself to do some more examples of how useful this stuff can get and because I want to get guide this OUT, damn it. I was about to do a King-Of-The-Hill clone with a friend of mine called “Capture the broccoli”, but since life is messy, this might never see daylight.

I talked to various people, even tried to get ydnar (;) , and got mixed feelings by some of them.

Yes indeed, this is horribly complex and might be too much work for what you get out of it, but as I said in the intro, it has been a fun ride for me and I got so much out of it.

So please excuse me if this guide doesn't help you at all, and be my guest if you want to help me working on it !

wviperw: i mean more applications

--skOre--: yes sure

wviperw: saying that it can be used for a door in which you have to shoot 4 buttons--i'd call that an example

wviperw: saying that you can use it in your coop mod so that the four players on the team have to separately fight their way to 4 different buttons so they can open the master door to the level boss--i'd call that an application

wviperw: you need more applications :)

--skOre--: Yes, believe me

--skOre--: Did you ever played Jedi Knight ?

--skOre--: the old ones

--skOre--: the really old

wviperw: dark forces?

--skOre--: where you fly through tunnels and have to shoot buttons in order to open doors you fly through

wviperw: nope

--skOre--: that would be possible

wviperw: you can do that w/ regular q3... can't you?

--skOre=: I can, yes
--skOre=: with some electronics
--skOre=: ;)
wvipew: i mean w/o elect
--skOre=: no
--skOre=: If you want to say "shoot 32 buttons to open the door" then not
wvipew: oh :)
--skOre=: then you need an AND-Gate
wvipew: right
--skOre=: See, this opens a whole new dimension on quake3 editing
wvipew: how often do you need to shoot N buttons to open a door though?
--skOre=: As well possible
wvipew: 1 button is enough most of the time
--skOre=: Or, Shoot Button A four times, then button B three times and then A one time
again
--skOre=: You can have numpads where you have to input a number
--skOre=: I know its not relevant for hardcore DM or CPMA stuff
--skOre=: ;)
wvipew: hehe
--skOre=: But I feel a new breeze in q3 mapping would be a good thing
--skOre=: like mods-in-maps
wvipew: you can do a numpad?
--skOre=: Sure
--skOre=: I can do a calculator
--skOre=: But that would really need time ;)
--skOre=: As I said I also did layouts on FlipFlops that can store values
wvipew: do a quak3 game inside of quake3...
wvipew: :P
--skOre=: heh
--skOre=: Yes, its where q3 meets the philosophy of what is necessary ;)
--skOre=: The problem is MAX_ENTITIES = 1024
--skOre=: ;)
wvipew: yep
--skOre=: Maybe I could code you microprocessors with more
--skOre=: but...
wvipew: that'd be pointless
--skOre=: Off course ;)
wvipew: like trying to use a graphing calculator to do supercomputer work
--skOre=: heh yes
wvipew: hehe, you feelin' like a Matrix theme? :
wvipew: "The first step is to disregard everything you think you know about the world
around you which you call Radiant. Do this, you will learn just how
deep the rabbit hole goes..."

--skOre=: ^^
--skOre=: Yes
--skOre=: How very deep that hole goes ;)
--skOre=: Maybe I should feature some views of people like you, explaining what they think
is the point of all this
wvipew: why its to take over the world with machines
wvipew: the Matrix has you.
--skOre=: Like:"Hi, I'm wvipew, and if you do CPMA mapping, screw Electronics and try to
learn how to really layout a good map"

--skOre=: heh
wvipew: lol
--skOre=: Yes, that would rock ;)
wvipew: actually, i could see CPM TDM possibly using it
--skOre=: Yes
wvipew: Like you need 2 players to be at a spot in order for the trap door to quad to open
--skOre=: it has very limited sense in DM, but when it comes to team action, it really turns
out cool

--skOre=: yes
--skOre=: Or The teams have to hold gates open for a player trying to return a flag
wvipew: yeah
--skOre=: See? It got you as well ;P

**THIS DOCUMENT IS A FRIGGIN MESS !
KICK ME TO DO IT BETTER ;)**

X - LINKS

Must Reads:

Q3Radiant Editor Manual (<http://www.qeradiant.com/manual/>)

Forums:

Quake3world LE&M Forum: (<http://www.quake3world.com/cgi-bin/forumdisplay.cgi?action=topics&forum=Level+Editing+|AMP|+Modeling&number=6>)

X - THANKS

Big thanks out to:

4days (for the initial idea for silent func_doors)

ydnar (oh how can a mapper be more thankful to anybody ;))

wvipew (for checking on my spelling, and for the general "WHY?" idea, sorry that I didnt completely worked this baby over as I should have with your correction ...)

X – LEGAL STUFF

David 'skOre' Deutsch is author of this document, thus all copyright (2003) belongs to him.

If you like this idea and want to use it in your own maps, give me credit in your readme containing a link to my webpage (<http://www.skore.de>) and my email address (skOre@skore.de), and some words to pimp me (;-)). I might be no big claimer on copyrights though, but I want you to use your brains and work with ME if you have ideas on it

This document may be electronically distributed only at NO CHARGE to the recipient in its current state, it has to be complete (the whole .zip-file containing all the .maps and .pk3's) and unaltered.

UNDER NO CIRCUMSTANCES IS THIS DOCUMENT TO BE DISTRIBUTED ON A DATA CARRIER (DVD/CD/HD etc.) WITHOUT PRIOR WRITTEN PERMISSION.

Quake III Arena is a registered trademark of id Software, Inc.

This Document was created with OpenOffice.org 1.1.0

All my love to the Open Source Community!
