From

http://quakeone.com/forum/quake-help/general-help/8364-become-a-quake-modder/page2

Search and Rescue (super tut)

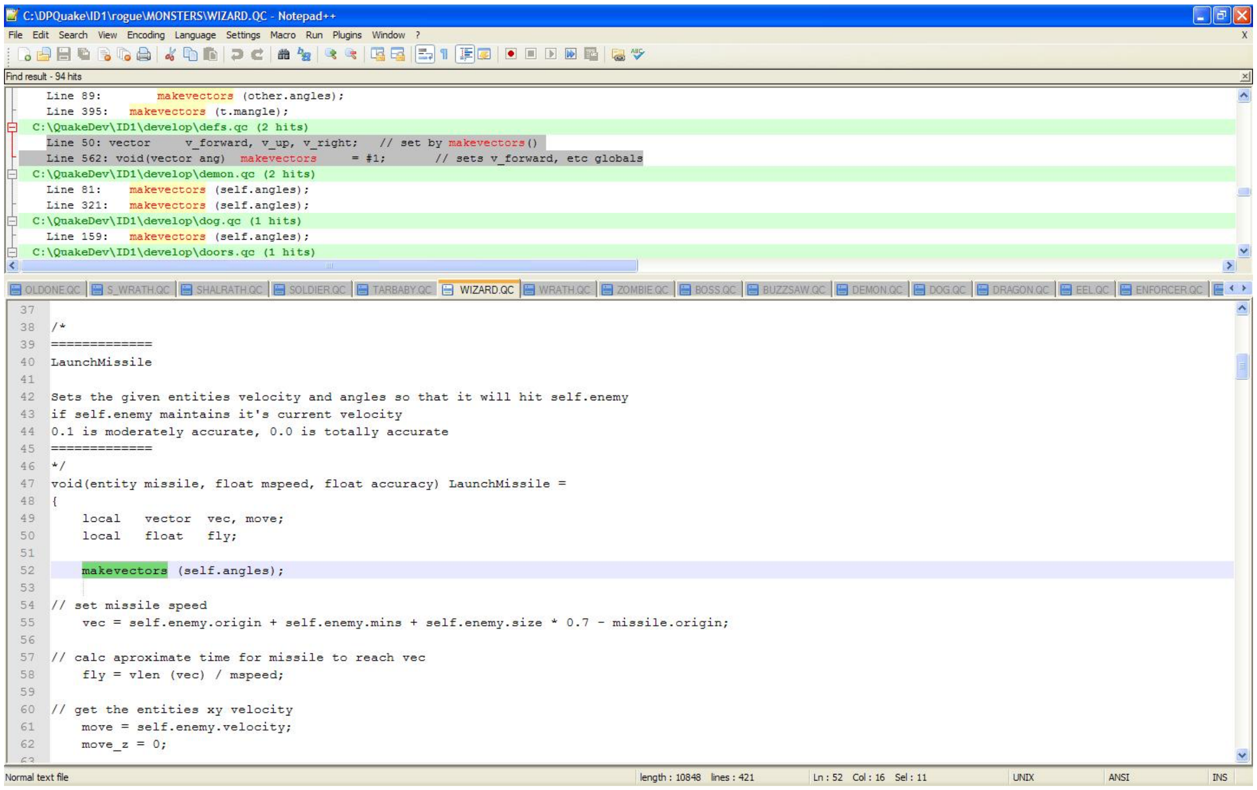
Search and Rescue (super tut) rev 2.0

Alright, I'm gonna do something a little different this time. I am going to teach you how to learn. I know that sounds retarded, but by the end of this you will not think this is retarded at all. It's supposedly a "fact" that you have to tell someone something 7 times before they officially commit it to memory, so, I'm going to make this tutorial 7 repetitions.

First of all, I am using Notepad++ for this tutorial. It's free and it's awesome. You should go get it so you can have this invaluable tool at your disposal.

In notepad++ I opened every single .QC that I have, all at one time. I then randomly clicked a tab and "threw" the scrollbar. I wound up on the LaunchMissle function in Wizard.qc. I then started at the top of the function and read until I hit something I didn't know (it didn't take long).

Research level: 1


C:\DPQuake\ID1\rogue\MONSTERS\WIZARD.QC - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Find result - 94 hits
OLDONE.QC S_WRATH.QC SHALRATH.QC SOLDIER.QC TARBABY.QC WIZARD.QC WRATH.QC ZOMBIE.QC BOSS.QC BUZZSAW.QC DEMON.QC DOG.QC DRAGON.QC EEL.QC ENFORCER.QC
Normal text file
length : 10848 lines : 421
Ln : 52 Col : 16 Sel : 11
UNIX
ANSI
INS

```
    Line 89:        makevectors (other.angles);
    Line 395:   makevectors (t.mangle);
  C:\QuakeDev\ID1\develop\defs.qc (2 hits)
    Line 50: vector     v_forward, v_up, v_right;   // set by makevectors()
    Line 562: void(vector ang)  makevectors      = #1;        // sets v_forward, etc globals
  C:\QuakeDev\ID1\develop\demon.qc (2 hits)
    Line 81:    makevectors (self.angles);
    Line 321:   makevectors (self.angles);
  C:\QuakeDev\ID1\develop\dog.qc (1 hits)
    Line 159:   makevectors (self.angles);
  C:\QuakeDev\ID1\develop\doors.qc (1 hits)
```

```

/*
=============
LaunchMissile

Sets the given entities velocity and angles so that it will hit self.enemy
if self.enemy maintains it's current velocity
0.1 is moderately accurate, 0.0 is totally accurate
=============
*/
void(entity missile, float mspeed, float accuracy) LaunchMissile =
{
    local   vector  vec, move;
    local   float   fly;

    makevectors (self.angles);

// set missile speed
    vec = self.enemy.origin + self.enemy.mins + self.enemy.size * 0.7 - missile.origin;

// calc aproximate time for missile to reach vec
    fly = vlen (vec) / mspeed;

// get the entities xy velocity
    move = self.enemy.velocity;
    move_z = 0;
```

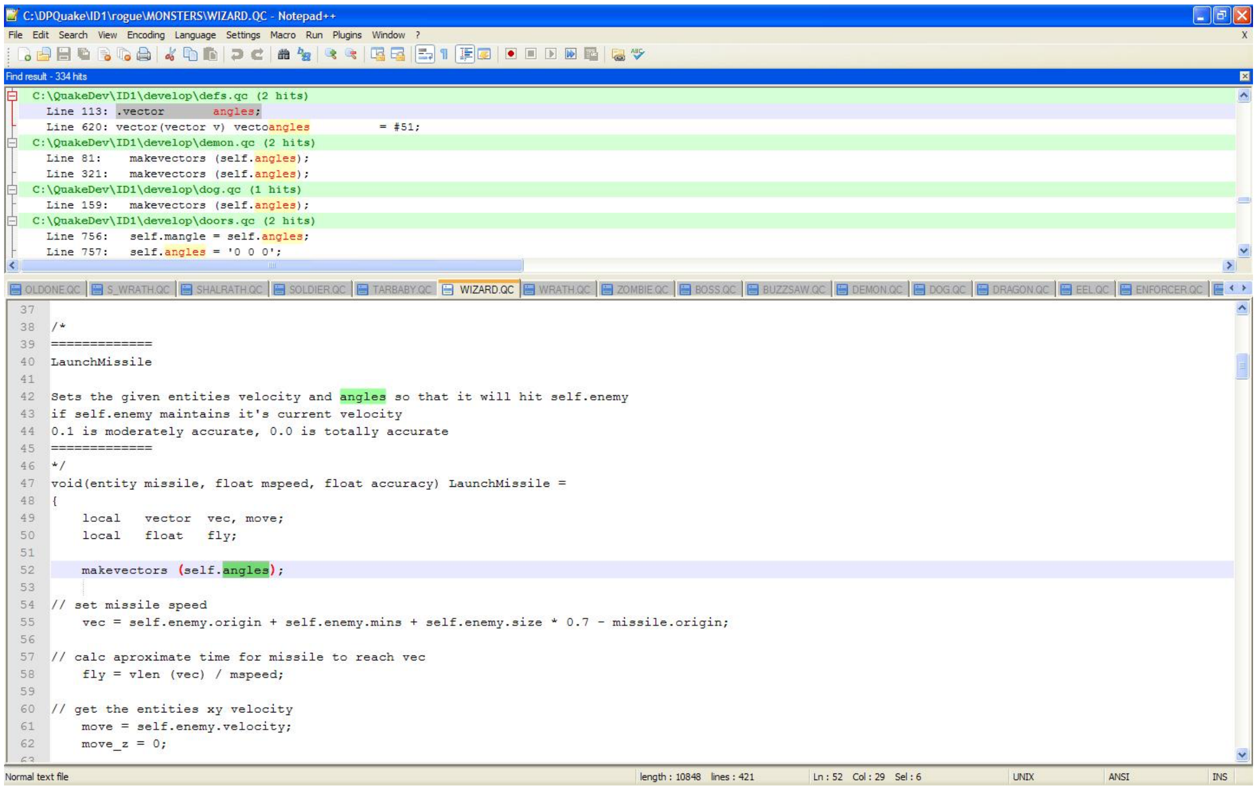
As you can see I have *makevectors* highlighted in the image. Of course, I understand what something so accurately named is doing, but I want to know how it works. So, I highlighted it, clicked search (in the top bar) and then chose find. Notepad++ automatically places my selection in the find box, so I just clicked *find all in all opened documents*. The results were what you see at the top of the image.

We get a nice little chunk of info. From what I highlighted in grey, we can see that makevector sets v_forward, v_up and v_right, but just as important - those three vars are not field types (there is no dot before the vector type). So this means that these vars are global and not a declared property of any specific entity.

However, there is even more info in that little gray hilight. You see how makevectors = #1. This means that makevectors is a built-in function. In other words, make vectors is **used** by the quake c code, but it is processed and defined in whatever engine you use. Not only is it a built-in function, but it is the first one. Which actually means little more than you are learning them in order. (lol)

Let's go ahead and polish off our line by doing a *find all in all opened documents* for angle and see what else it can tell us

Research level: 2



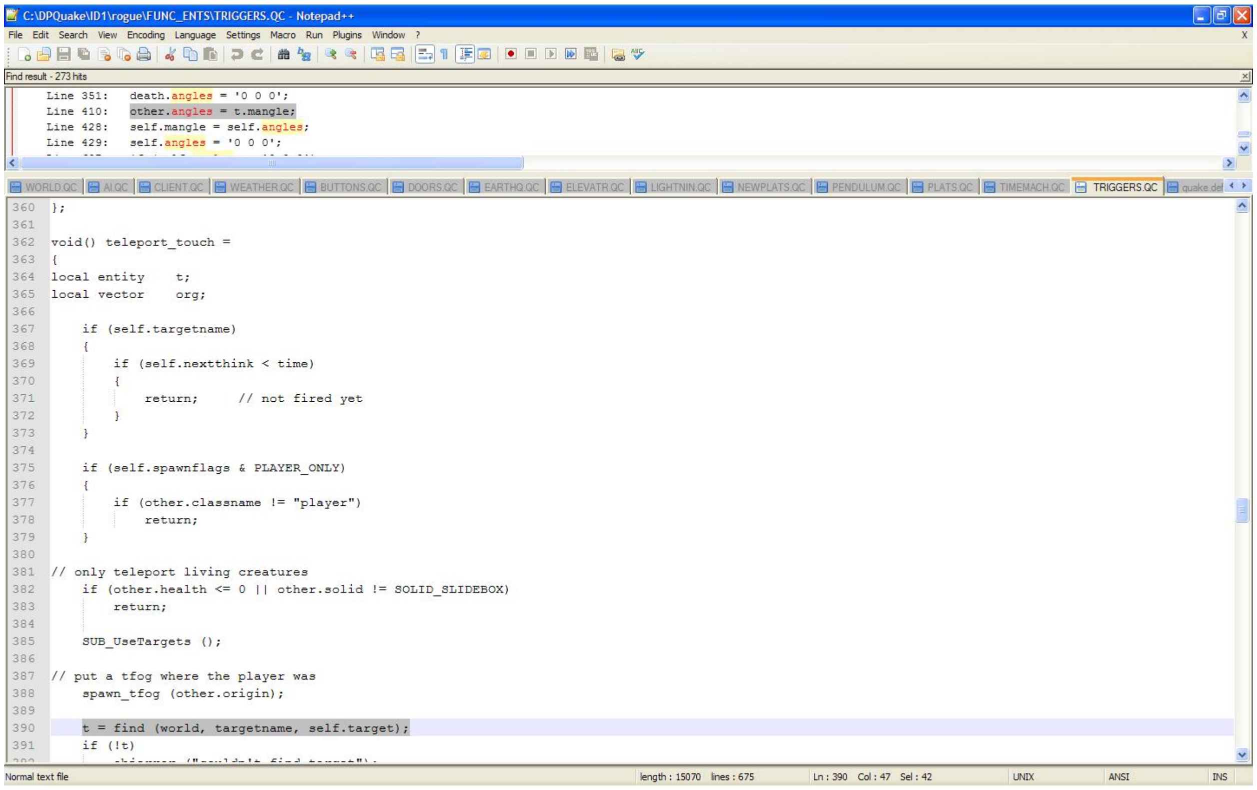
```
Find result - 334 hits
C:\QuakeDev\ID1\develop\defs.qc (2 hits)
  Line 113: .vector        angles;
  Line 620: vector(vector v) vectoangles          = #51;
C:\QuakeDev\ID1\develop\demon.qc (2 hits)
  Line 81:    makevectors (self.angles);
  Line 321:   makevectors (self.angles);
C:\QuakeDev\ID1\develop\dog.qc (1 hits)
  Line 159:   makevectors (self.angles);
C:\QuakeDev\ID1\develop\doors.qc (2 hits)
  Line 756:   self.mangle = self.angles;
  Line 757:   self.angles = '0 0 0';
```

```

/*
=============
LaunchMissile

Sets the given entities velocity and angles so that it will hit self.enemy
if self.enemy maintains it's current velocity
0.1 is moderately accurate, 0.0 is totally accurate
=============
*/
void(entity missile, float mspeed, float accuracy) LaunchMissile =
{
	local	vector	vec, move;
	local	float	fly;

	makevectors (self.angles);

// set missile speed
	vec = self.enemy.origin + self.enemy.mins + self.enemy.size * 0.7 - missile.origin;

// calc aproximate time for missile to reach vec
	fly = vlen (vec) / mspeed;

// get the entities xy velocity
	move = self.enemy.velocity;
	move_z = 0;
```

Well, there isn't a whole lot there. So, lets take this research level to address a few other things. When I do the document-wide searches, I am scrolling through the results looking for DEFS.qc. This is the .qc where the grand majority of Quakes variables are defined. I don't just rush to DEFS.qc though. I scan all the other results on the way and fill my head up with my search's various implementations.

Scanning, that is a nice segue to the next thing we will address. Scan down to the bottom of the results and you will see self.mangle = self.angles. Mangle eh, what the hell is that? Let's find out. First off, we can actually click that line in the search results and it will bring us directly to that line of code in the .qc class that it resides in. So, that's what I did. Do I need to tell you that, once I was brought to the line, I hilighted mangle, clicked search and selected find - followed by clicking find all in all opened documents? Well, I just did.

Research level: 3


C:\DPQuake\ID1\rogue\FUNC_ENTS\TRIGGERS.QC - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Find result - 273 hits
WORLD.QC AI.QC CLIENT.QC WEATHER.QC BUTTONS.QC DOORS.QC EARTHQ.QC ELEVATR.QC LIGHTNIN.QC NEWPLATS.QC PENDULUM.QC PLATS.QC TIMEMACH.QC TRIGGERS.QC
Normal text file length : 15070 lines : 675 Ln : 390 Col : 47 Sel : 42 UNIX ANSI INS

```
Line 351:   death.angles = '0 0 0';
Line 410:   other.angles = t.mangle;
Line 428:   self.mangle = self.angles;
Line 429:   self.angles = '0 0 0';
```

```
};

void() teleport_touch =
{
local entity    t;
local vector    org;

    if (self.targetname)
    {
        if (self.nextthink < time)
        {
            return;     // not fired yet
        }
    }

    if (self.spawnflags & PLAYER_ONLY)
    {
        if (other.classname != "player")
            return;
    }

// only teleport living creatures
    if (other.health <= 0 || other.solid != SOLID_SLIDEBOX)
        return;

    SUB_UseTargets ();

// put a tfog where the player was
    spawn_tfog (other.origin);

    t = find (world, targetname, self.target);
    if (!t)
```

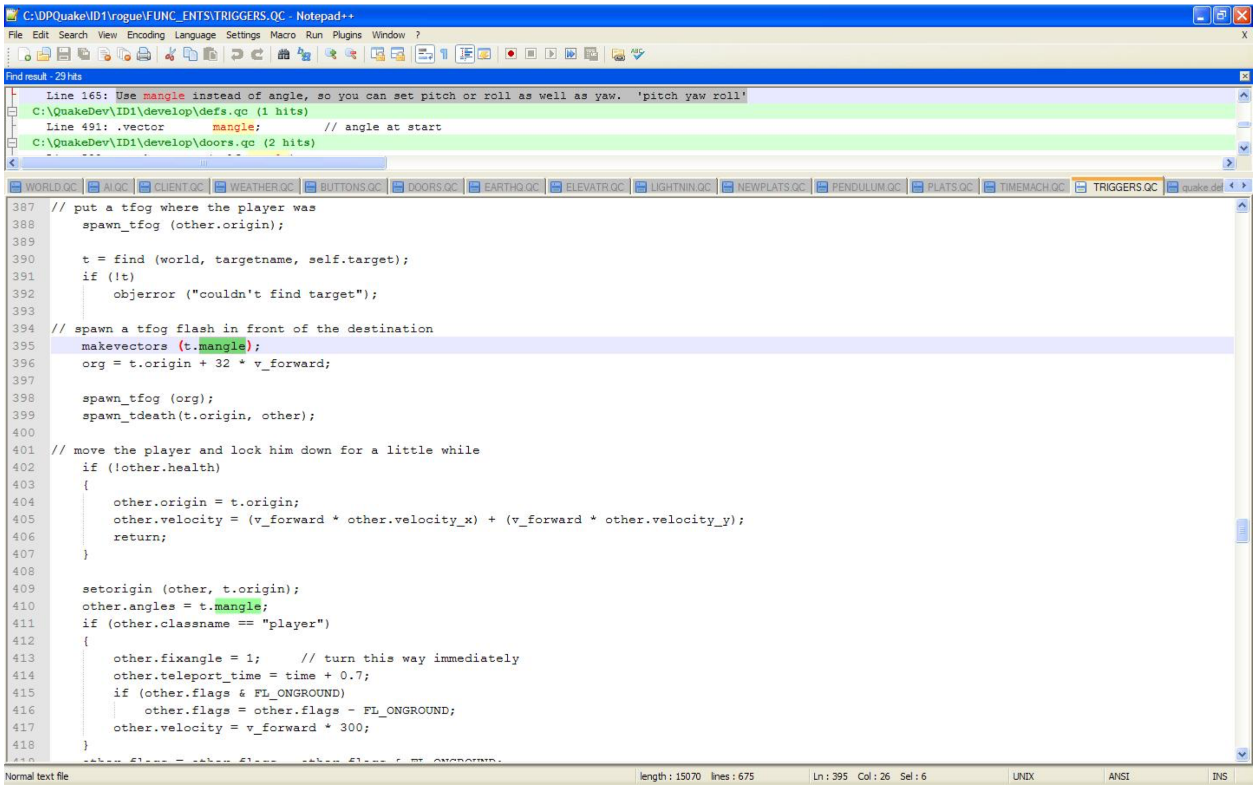
When the search results appeared, on my way to DEFS.qc, I came across this line (hilighted in the results). It piqued my interest and I found myself in TRIGGERS.qc (this is kinda like a nerd adventure...right? LOL). Now, I already knew that "t" was going to be an entity, because it has a field property. Only entities have field types. However, I wanted to know - what is this "t" they speak of.

Well, first off the function is teleport_touch. Can you guess what this function is for o.0? The first occurrence of "t" (aside from being declared a local entity) is when it is being assigned the return value of the "find" method. Obviously "find" is going to return an entity, but there is something very strange here. If you scan up you will see a line that is wrapping self.targetname in an if statement. This is because targetname is a field property of an entity. To make it simple - it is the entity's unique name*. The problem is, in our find function, targetname is not joined to an entity.

Strange indeed we will have to remember to research this at another level. For now though, we still need to figure out what a mangle is. Off to DEFS.qc.

*unique name/targetname: when you build maps, in order to determine one object from another, you give entities unique names. These names can be anything the mapper wants them to be. Their sole purpose is to distinguish them from other entities of the same type. So, let's assume that you have a trigger that opens a door. Well you don't want the trigger to open every door, so you give the door a targetname and then put that name in the triggers target field. That's how entities are connected and "talk" to each other.

Research level: 4


C:\DPQuake\ID1\rogue\FUNC_ENTS\TRIGGERS.QC - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Find result - 29 hits
WORLD.QC AI.QC CLIENT.QC WEATHER.QC BUTTONS.QC DOORS.QC EARTHQ.QC ELEVATR.QC LIGHTNIN.QC NEWPLATS.QC PENDULUM.QC PLATS.QC TIMEMACH.QC TRIGGERS.QC quake.def
Normal text file length : 15070 lines : 675 Ln : 395 Col : 26 Sel : 6 UNIX ANSI INS

```
    Line 165: Use mangle instead of angle, so you can set pitch or roll as well as yaw.  'pitch yaw roll'
  C:\QuakeDev\ID1\develop\defs.qc (1 hits)
    Line 491: .vector       mangle;         // angle at start
  C:\QuakeDev\ID1\develop\doors.qc (2 hits)
```

```
// put a tfog where the player was
    spawn_tfog (other.origin);

    t = find (world, targetname, self.target);
    if (!t)
        objerror ("couldn't find target");

// spawn a tfog flash in front of the destination
    makevectors (t.mangle);
    org = t.origin + 32 * v_forward;

    spawn_tfog (org);
    spawn_tdeath(t.origin, other);

// move the player and lock him down for a little while
    if (!other.health)
    {
        other.origin = t.origin;
        other.velocity = (v_forward * other.velocity_x) + (v_forward * other.velocity_y);
        return;
    }

    setorigin (other, t.origin);
    other.angles = t.mangle;
    if (other.classname == "player")
    {
        other.fixangle = 1;     // turn this way immediately
        other.teleport_time = time + 0.7;
        if (other.flags & FL_ONGROUND)
            other.flags = other.flags - FL_ONGROUND;
        other.velocity = v_forward * 300;
    }
```

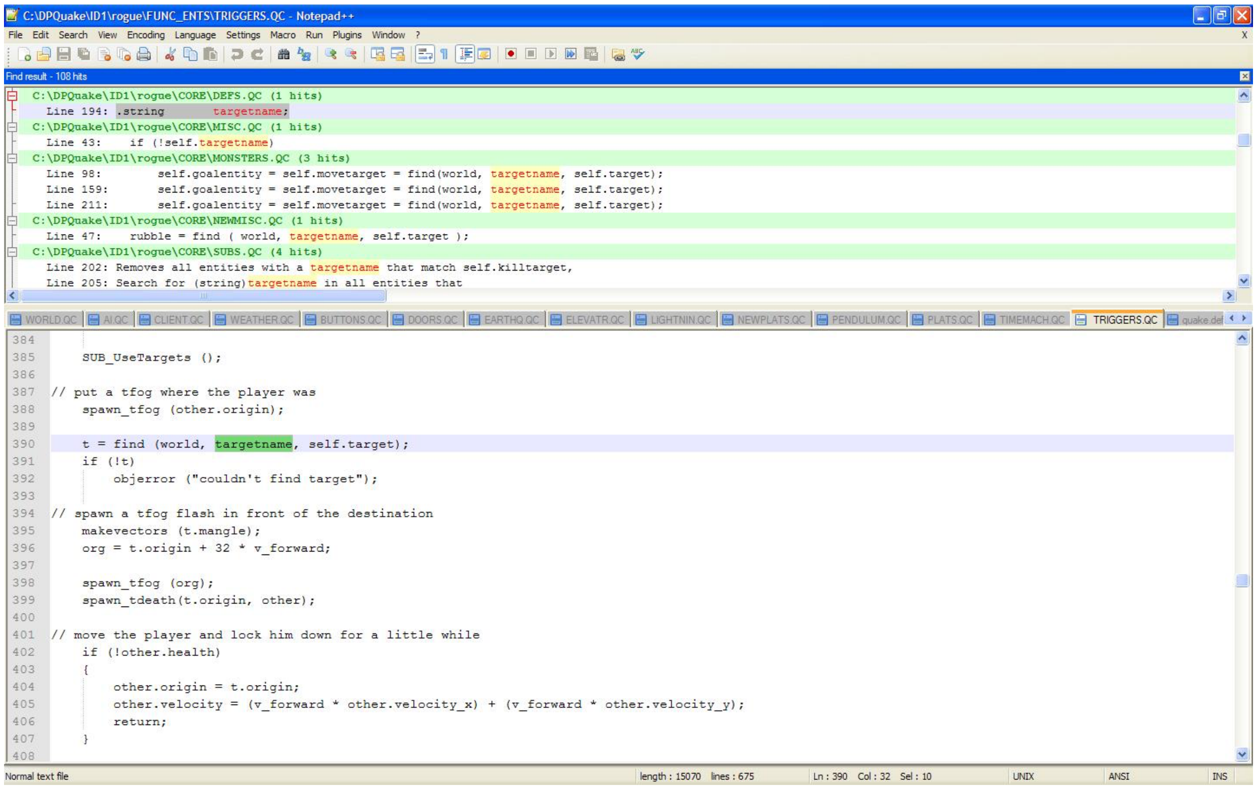
Jackpot! Man, ya' gotta love well commented code. It's right there above the mangle declaration. Mangle holds the variables for pitch, yaw and roll. Since that was so quick and awesome that it stole my thunder for this research level, let’s have a chit chat about vectors. So far, we have come across quite a few varieties of vectors and vector processors. It all started with makevectors, which takes the players current angles and sets the global vars v_forward, v_up and v_right. We have to assume that these vars are just place holders that are ready for any entity that needs to use them.

We have also come across mangle which **sets** the pitch yaw and roll. There is another often-used field type vector named origin. Now, vectors have a neat little feature. You can capture any property of a vector with what I call underscore syntax. Let's take angles for instance, we can capture the x property of angles with angles_x. The same goes for the y and z property. Origin and other vectors can be treated the same, EXCEPT mangle!

We got our answer on mangle really quick, but what the search results didn't show is the line directly above it. In this case (as it turns out) mangle is used to set the pitch yaw and roll of the INTERMISSION CAMERA. Now, this isn't it's only use, after all, we were in TRIGGERS.qc and mangle was being calculated from info_teleport_destination -oops you weren't supposed to know that yet. Forget I said that. Crap! now you are probably calculating the use of pitch yaw and roll from the teleport to the teleport destination. After all you don't want to teleport laying down...er I mean forget I said all of this 😉.

Upon further research of mangle you find that only some doors, platforms and a few monsters use it (mostly). Anyway, enough about mangle and vectors, we know what they do, They are orientation and position co-ordinates. We have that "find" method that we never researched hanging over our head. Let's use our handy-dandy document-wide search to see what we can learn

Research level: 5


C:\DPQuake\ID1\rogue\FUNC_ENTS\TRIGGERS.QC - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Find result - 108 hits
WORLD.QC AI.QC CLIENT.QC WEATHER.QC BUTTONS.QC DOORS.QC EARTHQ.QC ELEVATR.QC LIGHTNIN.QC NEWPLATS.QC PENDULUM.QC PLATS.QC TIMEMACH.QC TRIGGERS.QC quake.def
Normal text file
length : 15070 lines : 675
Ln : 390 Col : 32 Sel : 10
UNIX
ANSI
INS

```
C:\DPQuake\ID1\rogue\CORE\DEFS.QC (1 hits)
  Line 194: .string        targetname;
C:\DPQuake\ID1\rogue\CORE\MISC.QC (1 hits)
  Line 43:    if (!self.targetname)
C:\DPQuake\ID1\rogue\CORE\MONSTERS.QC (3 hits)
  Line 98:        self.goalentity = self.movetarget = find(world, targetname, self.target);
  Line 159:       self.goalentity = self.movetarget = find(world, targetname, self.target);
  Line 211:       self.goalentity = self.movetarget = find(world, targetname, self.target);
C:\DPQuake\ID1\rogue\CORE\NEWMISC.QC (1 hits)
  Line 47:    rubble = find ( world, targetname, self.target );
C:\DPQuake\ID1\rogue\CORE\SUBS.QC (4 hits)
  Line 202: Removes all entities with a targetname that match self.killtarget,
  Line 205: Search for (string)targetname in all entities that
```

```

    SUB_UseTargets ();

// put a tfog where the player was
    spawn_tfog (other.origin);

    t = find (world, targetname, self.target);
    if (!t)
        objerror ("couldn't find target");

// spawn a tfog flash in front of the destination
    makevectors (t.mangle);
    org = t.origin + 32 * v_forward;

    spawn_tfog (org);
    spawn_tdeath(t.origin, other);

// move the player and lock him down for a little while
    if (!other.health)
    {
        other.origin = t.origin;
        other.velocity = (v_forward * other.velocity_x) + (v_forward * other.velocity_y);
        return;
    }

```

You still here? Look at you researchin' and learnin' and stuff. What, you think you're smart or somethin'? (LOL). For real though, good job! OK, the find method. When we last left off with this there was an interesting syndrome where we were sticking a naked field type inside of the find method interface. Let's see if we can figure out why.

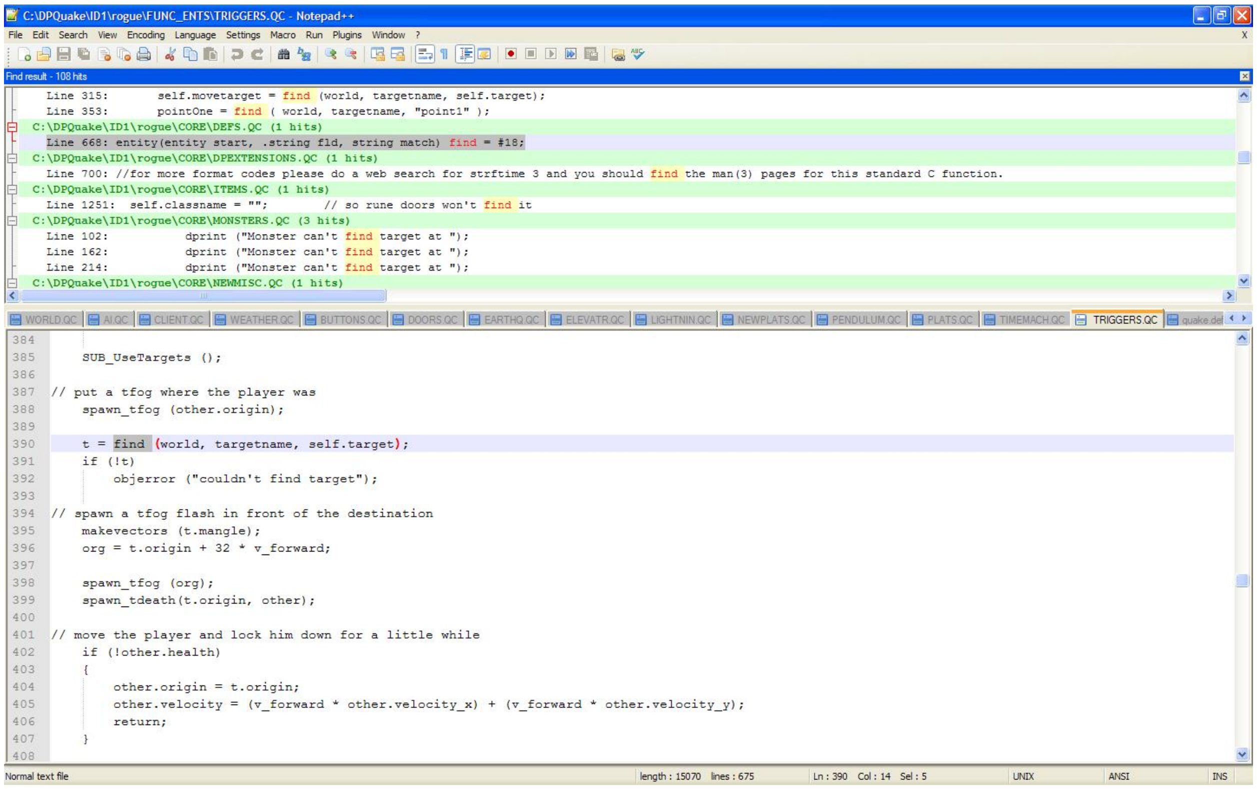
Well, a search of targetname just confirms that it is a field type. But look, even more find functions are using the naked version. Let's see if we can figure this out without cheating. Let's review some facts:

1) the first argument in the find method was world
2) the second argument was targetname which is the unique name of an entity.
3) the third argument was self.target, which is the name of the entity that self is targeting.
4) In this case we know that teleport_touch is self and (because I spilled the beans) we know that "t" is meant to equal the destination based on the return entity of find

Let's try and turn the statement into English instead of code. Find in the World the targetname that matches the name of self.target. Woah, we did it! That's the key. In this case targetname is not referring to any certain name, it is telling find to search all the targetnames for the one that matches self.target. We're awesome!

Let's see if we are right. Time for another document-wide search.

Research level: 6


C:\DPQuake\ID1\rogue\FUNC_ENTS\TRIGGERS.QC - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Find result - 108 hits
WORLD.QC AI.QC CLIENT.QC WEATHER.QC BUTTONS.QC DOORS.QC EARTHQ.QC ELEVATR.QC LIGHTNIN.QC NEWPLATS.QC PENDULUM.QC PLATS.QC TIMEMACH.QC TRIGGERS.QC quake.def
Normal text file
length : 15070 lines : 675
Ln : 390 Col : 14 Sel : 5
UNIX
ANSI
INS

```
  Line 315:       self.movetarget = find (world, targetname, self.target);
  Line 353:       pointOne = find ( world, targetname, "point1" );
C:\DPQuake\ID1\rogue\CORE\DEFS.QC (1 hits)
  Line 668: entity(entity start, .string fld, string match) find = #18;
C:\DPQuake\ID1\rogue\CORE\DPEXTENSIONS.QC (1 hits)
  Line 700: //for more format codes please do a web search for strftime 3 and you should find the man(3) pages for this standard C function.
C:\DPQuake\ID1\rogue\CORE\ITEMS.QC (1 hits)
  Line 1251:  self.classname = "";        // so rune doors won't find it
C:\DPQuake\ID1\rogue\CORE\MONSTERS.QC (3 hits)
  Line 102:           dprint ("Monster can't find target at ");
  Line 162:           dprint ("Monster can't find target at ");
  Line 214:           dprint ("Monster can't find target at ");
C:\DPQuake\ID1\rogue\CORE\NEWMISC.QC (1 hits)
```

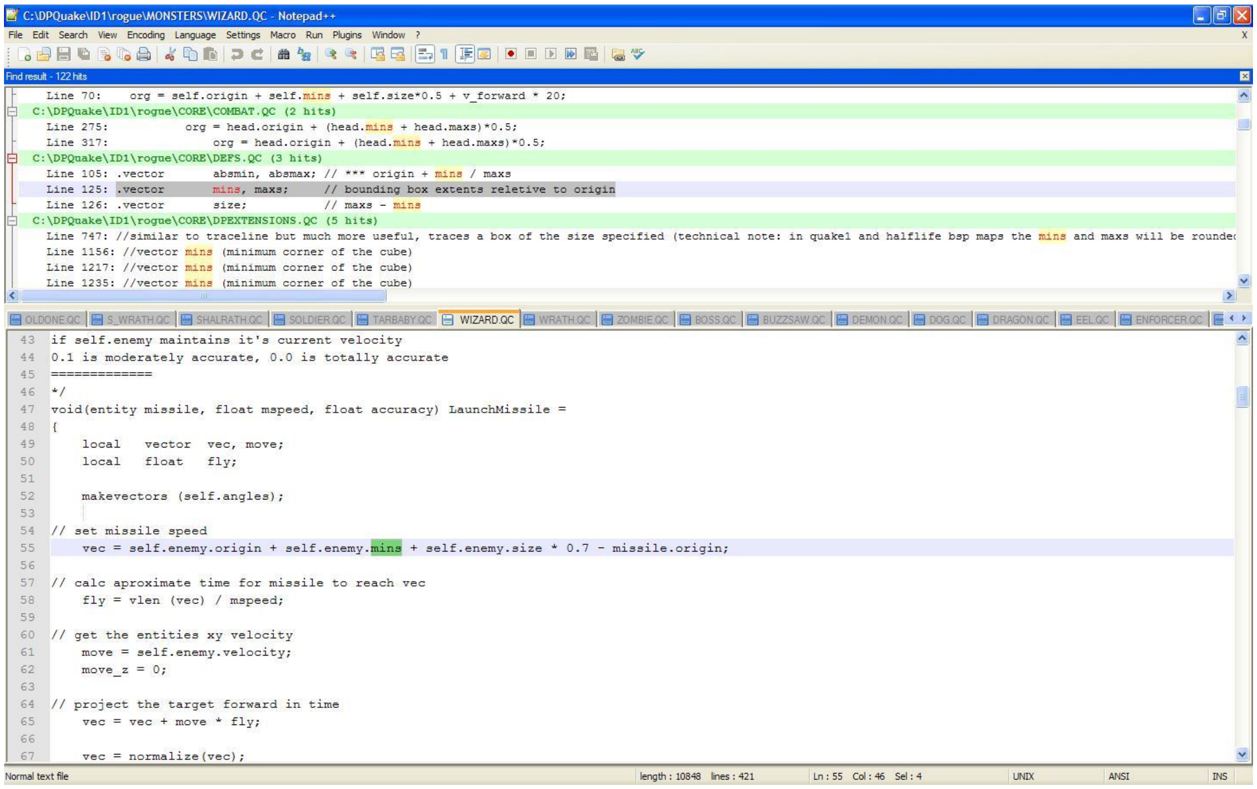
```

    SUB_UseTargets ();

// put a tfog where the player was
    spawn_tfog (other.origin);

    t = find (world, targetname, self.target);
    if (!t)
        objerror ("couldn't find target");

// spawn a tfog flash in front of the destination
    makevectors (t.mangle);
    org = t.origin + 32 * v_forward;

    spawn_tfog (org);
    spawn_tdeath(t.origin, other);

// move the player and lock him down for a little while
    if (!other.health)
    {
        other.origin = t.origin;
        other.velocity = (v_forward * other.velocity_x) + (v_forward * other.velocity_y);
        return;
    }

```

Yup! Look, the second parameter of find accepts a string. in this case targetname is not a string but it is a field type of string. Look, they even named the var fld (field) AND they used the dot syntax to declare that only a field type string can be used. Targetname is a field type string. With more scanning of the search results we find there are other field type strings that are used. Classname is one. Classname is the type of entity something is (monster, door, light, etc). So, in that case it would return a certain kind of entity as opposed to a specifically named one.

So, that's a good lesson, right? We learned that you don't always have to put the information of a var in a methods interface. Sometimes it can be simply a var of the correct type and within the method it will be used as a comparison to find things. Like a pointer. Another thing we can learn from these search result is that "find" is a built-in function, but it's not the second one, so I guess our learning them in order thing was short lived.

Well, we have actually hit a dead end. Or have we? Do you know everything about qc yet? Me either. You know what we do when we reach this point? We go back to the beginning and work our way down the first function where we left off, til we find something that we don't understand. Back to the LaunchMissle method in WIZARD.qc

Research level: 7



```
C:\DPQuake\ID1\rogue\MONSTERS\WIZARD.QC - Notepad++
Find result - 122 hits
   Line 70:    org = self.origin + self.mins + self.size*0.5 + v_forward * 20;
 C:\DPQuake\ID1\rogue\CORE\COMBAT.QC (2 hits)
   Line 275:           org = head.origin + (head.mins + head.maxs)*0.5;
   Line 317:               org = head.origin + (head.mins + head.maxs)*0.5;
 C:\DPQuake\ID1\rogue\CORE\DEFS.QC (3 hits)
   Line 105: .vector       absmin, absmax; // *** origin + mins / maxs
   Line 125: .vector       mins, maxs;     // bounding box extents reletive to origin
   Line 126: .vector       size;           // maxs - mins
 C:\DPQuake\ID1\rogue\CORE\DPEXTENSIONS.QC (5 hits)
   Line 747: //similar to traceline but much more useful, traces a box of the size specified (technical note: in quake1 and halflife bsp maps the mins and maxs will be roundec
   Line 1156: //vector mins (minimum corner of the cube)
   Line 1217: //vector mins (minimum corner of the cube)
   Line 1235: //vector mins (minimum corner of the cube)
```

```
if self.enemy maintains it's current velocity
0.1 is moderately accurate, 0.0 is totally accurate
=============
*/
void(entity missile, float mspeed, float accuracy) LaunchMissile =
{
    local   vector  vec, move;
    local   float   fly;

    makevectors (self.angles);

// set missile speed
    vec = self.enemy.origin + self.enemy.mins + self.enemy.size * 0.7 - missile.origin;

// calc aproximate time for missile to reach vec
    fly = vlen (vec) / mspeed;

// get the entities xy velocity
    move = self.enemy.velocity;
    move_z = 0;

// project the target forward in time
    vec = vec + move * fly;

    vec = normalize(vec);
```

```
Normal text file    length : 10848   lines : 421    Ln : 55   Col : 46   Sel : 4    UNIX    ANSI    INS
```

LOL! Of course, our search for something to know would start at practically the next line. What the hell is a mins? Well, according to my document-wide search...........

Epilogue

Well, using nothing but an advanced text editor, curiosity and our brain, we were able to take a little journey through the Quake source code and decipher many of its mysteries. Personally, I feel like this method, coupled with using the QC manual is the best way to learn QC.

QC is not a very advanced or complicated language. In many ways, it isn't even as versatile or rich as Javascript. What makes QC confusing is its ambiguous scope and unconventional syntax. Once you learn to decipher the ambiguity and commit the syntax to memory, you will fly as a QC modder.

Thank you for sticking with me for this tutorial,
Gypsy

Last edited by MadGypsy; 08-25-2017, 09:54 AM.