

## Homing missiles in Quake III Arena

### Introduction

In this short tutorial, i'll add a new type of rockets that can see and follow the surrounding enemies. The user will be able to switch from the normal mode to the new one at any time during the game. In addition, the rockets will ignores the players that are covered by obstacles such as walls.

### Firing mode

The received client commands are handled by the server by a function named ClientCommand, declared in the g\_cmds.c source file.

```
void ClientCommand( int clientNum )
{
    ...
    ...
    ...

    else if (Q_stricmp (cmd, "stats") == 0)
        Cmd_Stats_f( ent );

    // If the received command is equal to "HomingMissilesMode" the code will call the
    // vSetHomingMissilesMode function
    else if (Q_stricmp(cmd, "HomingMissilesMode") == 0)
        vSetHomingMissilesMode(ent);

    else
        trap_SendServerCommand( clientNum, va("print \"unknown cmd %s\n\"", cmd ) );
}
```

With this code, when the user will write those command in his console (or by key binded to the command) the function i've declared will be called.

Now, let's declare the function, above the previous modified function.

```
//
// Sets or unsets the homing missiles firing mode for the rocket launcher
//
// Parameters
// genPlayer: It represents the player that have called the "HomingMissilesMode" command
//
// Return value: none
//

void vSetHomingMissilesMode(gentity_t *genPlayer)
{
```

```
// Change the firing mode variable (more about this later) to active if it was deactivated and vice-versa
genPlayer->client->pers.qbHomingMissilesMode = !genPlayer->client->pers.qbHomingMissilesMode;

// Prints a message to the player's screen
trap_SendServerCommand(genPlayer->s.clientNum, va("print \"Homing missiles mode: %d\n\"", genPlayer->client->pers.qbHomingMissilesMode));

return;
}
```

The `trap_SendServerCommand` function allows the programmer to send a command to the client; in this code, i simply print a message to its monitor, in order to tell to him that the firing mode has been changed. The first parameter is the client number, that you can obtain by reading the `Entity->s.clientNum` member; the second one its the string to send. The `va` function is very handy cause allows us to easily format a string.

The `pers` structure i'm accessing in the above code, is called `clientPersistant_t`, because its content is always mantained at its values between respawns.

I've to modify it in order to store the firing mode, as well as the current target the last missile has locked on.

Let's open the `g_local.h` file and add the two variables:

typedef struct

```
{
    clientConnected_t connected;

    ...

    ...

    ...

    // If equal to qtrue, the homing missiles firing mode is enabled
    qboolean qbHomingMissilesMode;

    // Last homing missile target
    int iClientNumber;
} clientPersistant_t;
```

This structure is automatically initialized every time a player joins the server, and every time the map is changed. To add the initialization code for my new variables, i need to modify the `ClientBegin` function declared in the `g_client.c` source file.

```
void ClientBegin( int clientNum )
{
    gentity_t *ent;
    gclient_t *client;
    gentity_t *tent;
    int flags;

    ent = g_entities + clientNum;
    client = level.clients + clientNum;

    // Set the variable to qfalse
    client->pers.qbHomingMissilesMode = qfalse;

    // Set the target to MAX_GENTITIES + 1
    client->pers.iClientNumber = MAX_GENTITIES + 1;

    ...
    ...
    ...
}
```

In order to complete the firing mode support, i only need to modify the `fire_rocket` function in the `g_missile.c` source file. This function is responsible for the rocket launcher firing key and i've to add the code that checks the `qbHomingMissilesMode` persistent variable.

```
gentity_t *fire_rocket (gentity_t *self, vec3_t start, vec3_t dir)
{
    gentity_t *bolt;

    VectorNormalize (dir);
    bolt = G_Spawn();
    bolt->classname = "rocket";

    // Check if the client has selected the homing missiles mode
    if (self->client->pers.qbHomingMissilesMode)
    {
        // Set the next think time to the next 500 msec in order to launch
        // an homing missile that will move like a normal one for the first half-second
        // At the end of this time, the engine will call my vThink_HomingMissile
        // callback, that will search for a valid target around itself
        bolt->nextthink = level.time + 500;
        bolt->think = vThink_HomingMissile;
    }
    // If the new firing mode is not selected, let's use the default values
    else
    {
        bolt->nextthink = level.time + 15000;
        bolt->think = G_ExplodeMissile;
    }
}
```

```
}  
  
...  
...  
...  
}
```

The nextthink variable, tells the engine the time when he have to call the think callback. The level structure is global, and its time member holds the current time.

```
//  
// Think callback for the homing missiles  
//  
// Parameters  
// genRocket: Holds the pointer to the rocket that have called the function  
//  
// Return value: none  
//
```

```
void vThink_HomingMissile(gentity_t *genRocket)
```

```
{  
  
    //  
    // Local variables  
    //  
  
    // Counter  
    int iLoop1;  
    // Last valid target found and its distance  
    int iTARGET, iDistance;  
    // Missile-target distance  
    vec3_t v3tDistance;  
    // Target position  
    vec3_t v3tTargetPosition;  
    // Missile direction  
    vec3_t v3tRocketAngles;  
    // Trace result (more about it later)  
    trace_t trResults;  
  
  
    //  
    // Code
```

```
//  
  
// If i've a target, i check if it's still active...  
if (g_entities[genRocket->parent->client->pers.iClientNumber].inuse)  
    genRocket->parent->client->pers.iClientNumber = MAX_GENTITIES + 1;  
  
// ...and that it's not dead  
if (g_entities[genRocket->parent->client->pers.iClientNumber].health <= 0)  
    genRocket->parent->client->pers.iClientNumber = MAX_GENTITIES + 1;  
  
// If i don't have any target, i need to serch for it  
if (genRocket->parent->client->pers.iClientNumber == MAX_GENTITIES + 1)  
{  
  
    // These variables tells which one of the last valid targets is the  
    // nearer to the missile  
    iTarget = MAX_GENTITIES + 1;  
    iDistance = 8192 * 16;  
  
    // Let's checks all the entities  
    for (iLoop1 = 0; iLoop1 < MAX_GENTITIES; iLoop1++)  
    {  
        // The slot must be active...  
        if (!g_entities[iLoop1].inuse)  
            continue;  
  
        // ...it must describe either a player or a bot...  
        if (!g_entities[iLoop1].client)  
            continue;  
  
        // ...it must not be the missile owner...  
        if (g_entities[iLoop1].s.clientNum == genRocket->parent->s.clientNum)  
            continue;  
  
        // ...it must be alive..  
        if (g_entities[iLoop1].health <= 0)  
            continue;  
  
        // ...and it must be within 500 units from the center of the rocket  
        VectorSubtract(genRocket->r.currentOrigin, g_entities[iLoop1].r.currentOrigin,  
v3tDistance);  
        if (VectorLength(v3tDistance) > 500)  
            continue;  
  
        // Now, i only need to check if it's not covered by an obstacle  
        trap_Trace(&trResults, genRocket->r.currentOrigin, NULL, NULL,  
g_entities[iLoop1].r.currentOrigin, genRocket - g_entities, MASK_SHOT);
```

```
// If the CONTENTS_SOLID is one, then there's a wall between the target
// and the rocket
if (trResults.contents & CONTENTS_SOLID)
    continue;

// This target could be valid, but i'll checks the remaining ones searching
// for a nearer target
if (VectorLength(v3tDistance) < iDistance)
{
    iDistance = VectorLength(v3tDistance);
    iTarget = iLoop1;
}
}

// If i've a valid target, i set it in the persistant client structure
if (iTarget != MAX_GENTITIES + 1)
{
    // Save the target
    genRocket->parent->client->pers.iClientNumber = iTarget;

    // Prints the message to the player's screen
    trap_SendServerCommand(genRocket->parent->s.clientNum, va("print \"Rilevato un
nemico!\n\""));
}
}

// If i've not targets, i'll explode the missile within 20 msec
if (genRocket->parent->client->pers.iClientNumber == MAX_GENTITIES + 1)
{
    // Next think time set to current time + 20 msec
    genRocket->nextthink = level.time + 20;

    // Explosion callback
    genRocket->think = G_ExplodeMissile;

    // Unset the target
    genRocket->parent->client->pers.iClientNumber = MAX_GENTITIES + 1;

    return;
}

// Get the target position
VectorCopy(g_entities[genRocket->parent->client->pers.iClientNumber].r.currentOrigin,
v3tTargetPosition);

// The position i've got actually refer to the target's feet; in order to point to its body, i need to
// add some units
v3tTargetPosition[2] += 10.0f;
```

```
// Get the orientation that points to the target
VectorSubtract(v3tTargetPosition, genRocket->r.currentOrigin, v3tRocketAngles);

// Set the new position to the rocket; you want, you can adjust its speed by normalizing
// v3tRocketAngles and scaling it by a factor with VectorScale
VectorCopy(genRocket->r.currentOrigin, genRocket->s.pos.trBase);
VectorCopy(v3tRocketAngles, genRocket->s.pos.trDelta);

// Next think time set to current time + 50 msec
genRocket->nextthink = level.time + 50;

return;
}
```

The VectorCopy, VectorScale, VectorSubtract, VectorNormalize, VectorCopy functions are in fact defined macros that handles the vectors operations. The vec3\_t data type is an array of three floats that holds the X, Y and Z coordinates.

Last step: if a rocket hits something, i need to remove the target i set in the persistent client structure. This event is handled by the G\_MissileImpact function, declared in the g\_missile.c file.

```
void G_MissileImpact( gentity_t *ent, trace_t *trace )
{
    gentity_t *other;

    ...
    ...
    ...

    // If it was an homing missile, unset the target variable
    if (ent->parent->client->pers.qbHomingMissilesMode)
        ent->parent->client->pers.iClientNumber = MAX_GENTITIES + 1;

    ...
    ...
    ...
}
```

Before ending the tutorial, i want to tell you that the g\_missile.c functions are not rocket launcher related; instead, they're general purpose functions that handles the events of any type of weapons that throw some type of bullet (just like the plasma gun).

The modifications are now completed, you only have to compile and test it, have fun.