

## Getting Started with Torque

(Based on Kevin Harris' Weapons tutorial: <http://www.codesampler.com/torque.htm>)

**Objectives** Upon finishing this exercise you should be able to:

1. Clone an existing game directory to start your own game.
2. Create a short-cut to run your game.
3. Use the in-game console to run arbitrary script commands.
4. Get a feel for the overall structure of Torque game assets.
5. Add models, textures, and scripts to a game.

**Resources** 1. Rocket Launcher mesh, scripts, etc.: <http://cs.potsdam.edu/faculty/laddbc/workshop34/RocketLauncher.zip>

### 1. Clone `tutorial.base` to `rocket.launcher`

Open the Windows Explorer. Navigate to the directory where you installed the Torque Game Engine (C:\Torque\TGE\_1\_5\_2 by default). Navigate down to `example`. Take note of the folders `starter.fps`, `starter.racing`, and `tutorial.base`. These three folders are the roots of sample games provided by GarageGames as part of the TGE. They are, unsurprisingly, a first-person shooter, a racer, and a simple tutorial.

One of the easiest ways to start a new game is to base it off of an existing game; these three games are yours to modify as you see fit so any one of them is a good place to start. We will start with `tutorial.base` because it is the simplest of the three folders.

Copy the `tutorial.base` folder in `example` and rename the copy to `rocket.launcher`.

### 2. Create a short-cut to run `rocket.launcher`.

All of the TGE demonstration games use the same executable program, `torqueDemo.exe` in the `example` folder where we copied the script folder. When starting the executable, the `-game` command-line switch specifies what script folder to use.

Open the TGE menu folder (C:\Documents and Settings\\Start Menu\Programs\Torque Game Engine 1.5 default) and make a copy of the `TutorialBase` short-cut. Rename the short-cut to `rocket.launcher`.

Open the properties on our new `rocket.launcher` short-cut. Notice that the target of the short-cut is `C:\Torque\TGE_1_5_2\example\torqueDemo.exe -game tutorial.base`. We just want to modify `tutorial.base` folder at the end of the target to our `rocket.launcher` folder.

### 3. Modify a script file (there is a typo in the shipping code).

Navigate to `example\rocket.launcher`. TorqueScript files use the `.cs` extension. The first file loaded by the engine is `main.cs`; we need to fix the name of the mission loaded when we tell it to load a mission.

Open `main.cs` in your text editor (we have been practicing with `emacs`). At the end of the file you will find the `loadMyMission` function. Line 112 should go from:

```
112 createServer("SinglePlayer", expandFilename("../data/missions/gameonemission.mis"));
```

to:

```
112 createServer("SinglePlayer", expandFilename("../data/missions/flat.mis"));
```

The original level file does not exist in the `tutorial.base` code so we have changed the mission to be the one mission that is defined in this game.

### 4. Using your short-cut, run the game; using the console, load your mission.

Run the game by double clicking on the short-cut you created. The game will come up in a menu mode with several icons across the top (it should look just like `tutorial.base`).

Click on the "Console" menu entry (or press the `~` key) to bring up the game console. A game console is a scripting shell. We can have the game load our mission by typing

```
loadMyMission();
```

and pressing `<Enter>`. Note that the Torque console has tab-completion (MSWindows style rather than `*nix` style) and it will append the semicolon if you don't include one.

Having typed this, the F World mission should load and you should fall onto a blue and white checkerboard-like surface. You can move around (using WASD keys) and you can toggle the third-person camera with the `<Tab>` key. Turning is done with the mouse.

Exit the game when you are done fooling around.

## 5. Permit turning and tilting by using the arrow keys.

Navigate to the `client` folder in `rocket.launcher`. Open the `default.bind.cs` file and navigate down to just past line 114, just after the definition of the `mouseTrigger` function.

`moveMap` is a keyboard map and the `bind` method permits binding of a key to an action. You can see the WASD movement keys and the mouse aiming:

```
116 moveMap.bind( keyboard, a, moveleft );
117 moveMap.bind( keyboard, d, moveright );
118 moveMap.bind( keyboard, w, moveforward );
119 moveMap.bind( keyboard, s, movebackward );
120 moveMap.bind( keyboard, space, jump );
121
122 moveMap.bind( mouse, xaxis, yaw );
123 moveMap.bind( mouse, yaxis, pitch );
124 moveMap.bind( mouse, button0, mouseTrigger );
```

We want to bind the arrow keys for looking in various directions and to bind the 'f' key for firing the weapon (no weapon yet, but soon).

```
116 moveMap.bind( keyboard, a, moveleft );
117 moveMap.bind( keyboard, d, moveright );
118 moveMap.bind( keyboard, w, moveforward );
119 moveMap.bind( keyboard, s, movebackward );
120 moveMap.bind( keyboard, space, jump );
121
122 moveMap.bind( keyboard, f, mouseTrigger );
123 moveMap.bind( keyboard, left, turnleft );
124 moveMap.bind( keyboard, right, turnright );
125 moveMap.bind( keyboard, up, panup );
126 moveMap.bind( keyboard, down, pandown );
127
128 moveMap.bind( mouse, xaxis, yaw );
129 moveMap.bind( mouse, yaxis, pitch );
130 moveMap.bind( mouse, button0, mouseTrigger );
```

Close and save `default.bind.cs`. If you want to see how turning and panning with the arrow keys works, run the game again, launch the mission, and have at it.

6. Add the `rocket_launcher` folder to the game.

Unzip the resource file for this lab. Inside the resource file there is the original `.htm` file for Kevin Harris' version of a weapons tutorial (for TGE 1.3) and the `rocket_launcher` folder. Inside the folder are folders for textures, shapes, sounds, and scripts.

Move the `rocket_launcher` folder to the `rocket_launcher` folder along side `client`, `data`, and `server`. All script changes assume that is where the file was put; you could, alternatively, move the contents of the various subfolders of the `assets` folder to their right place in the `data` folder.

7. Modify the server scripts to include `rocket_launcher.cs`.

Open up `server\game.cs` and find the definition of the `onServerCreated()` function. Add the `rocket_launcher` line *before* the other `exec` lines:

```
19 // Load up all datablocks, objects etc.
20 exec(expandFilename("~/rocket_launcher/scripts/rocket_launcher.cs"));
21 exec("./camera.cs");
22 exec("./editor.cs");
23 exec("./player.cs");
24 exec("./logoitem.cs");
```

(Actually, it is only required that the new `exec` come before `player.cs` because we are about to edit that file to give the player a rocket launcher.)

8. Modify `Player::onAdd` to add a rocket launcher to the player.

Open `server\player.cs` and find the definition of the `Player::onAdd` function. The body is, initially, just a comment between curly braces. Change it by adding a couple of lines:

```
35 function PlayerBody::onAdd(%this,%obj)
36 {
37 // Called when the PlayerData datablock is first...
38 %obj.mountImage( RocketLauncherImage, 0 );
39 %obj.setImageAmmo( 0, 1 );
40 }
```

The first line mounts the `RocketLauncherImage` (the rocket launcher mesh) and gives you ammunition. That way when you hit the ground, you have the weapon and can run around.

9. Fire up the game, start the mission, and look at how the weapon works.