Escher Player Physics
- A Quick(ish) Guide for Mappers

This is a quick overview of the new steps involved in making a map to work with the escher physics mutator. It assumes you already have a good understanding of unrealEd. I apologise in advance for the loose unstructured form of the document, it was compiled over many months as changes to the code were made, and I'm just too lazy to rewrite it.

This whole system relies almost entirely on custom pawn and playerController classes, and as such it's probably unlikely to be compatible with mods or mutators which use their own versions of these classes.

Before you Start

Before using anything, you will need to get the escher physics mutator embedded in your map. This makes it easy for players to play your map – they don't need to find the mutator and add it manually for the map to work properly.

You will firstly need the **escherPhysicsMut.u** and **escherPhysicsCode.u** files placed in your **UT2004/system** directory. Should be fairly straightforward. You must also add **escherPhysicsCode** to your editPackages and serverPackages in UT2004.ini. Note that it is completely unnecessary to add the **escherPhysicsMut** package to your serverPackages or editPackages lines, and in fact doing so will cause problems for you.

Open up unrealEd to the map you're working on or a new one, whatever. Type the following into the command entry field at the bottom left of your viewports:

```
obj load file=escherPhysicsMut.u package=myLevel
```

This will now have loaded up the mutator for your use in the map. If you only plan on doing this once, you can get rid of `escherPhysicsMut.u` file as you only need it for this step. Go to the actor class browser and uncheck and recheck 'show placeable classes only'. This is just to refresh the window. You are now able to see all the escher physics classes (if youre lucky, I find you usually need to type the command in at least twice to get it to actually load), so select **info/Mutator/escherMutTemplate/escherMut** and add it into the map wherever you like.

You're ready to go now – when someone loads up your map the mutator will already be running. Note that you **MUST** include the **escherPhysicsCode.u** file along with your map in order for it to run, and that you do not need **escherPhysicsMut.u** any more. Also make sure anyone who downloads your map to host multiplayer games with understands that they must add

```
serverPackages=escherPhysicsCode
```

to their UT2004.ini, or the map **will not load** for remote clients.

Things to Note

Some things that you would normally be able to do in an unreal level are not possible with the escher physics mutator running:

- The view will be very jumpy when walking over stairs or other small variations in the level. A good way to get around this is to use invisible staticMeshes or brushes (just use regular ones with an invisible texture) on your stairs to make the actual collision of them like a ramp. You cannot use blockingVolumes for this at the moment.
- Be careful with stairs, as players will only walk on surfaces that are roughly horizontal to the area they are currently in. So steep slopes might not be able to be walked on unless you add a volume pointing gravity more perpendicularly into them. There is a 45° tolerance each way.
- Players will slide over blockingvolumes without walking on them. Could be used to make some cool ice-like effects.
- Players sometimes get stuck on the joins between brushes, even if the join is perfectly flat. They also sometimes start to slide around a bit. You can't really do much about this, but you should be aware of it. Tapping the jump key makes the player unstuck/unslippery anyway.
- I've disabled the translocator for this mutator at the moment. It would do very weird things with reorienting the player after teleporting, which I am currently too lazy to fix.
- Pickups still fall with respect to normal level gravity if dropped.
- Doorways should be made wider than normal as the player is now the same width as height, for various reasons. The exact measurement is 88UU high and wide.
- Having two volumes with the same gravity actor touching each other can sometimes lead to problems, especially when also coinciding with a zone portal. If things start to go weird on you, I recommend either combining the brushes with the 2D shape editor, or making different gravity actors for each which point in an identical direction.
- Players sort of skip down slopes instead of walking flat on them.
- Bio Globs always fall downwards (relative to the world) when they drip from a surface.

To control gravity, you use a combination of zone gravity (`escherZones`) and volume gravity (`escherVolumes`).
Players decide which gravity to use by the following priorities:

- If neither in an `escherZone` or `escherVolume`, player will fall in the direction of their feet.
- If in an `escherZone` only, player uses the gravity vector of that zone.
- If in an `escherZone` and `escherVolume`, player uses the gravity of the volume.
- If in multiple `escherVolumes`, player uses the gravity of the `escherVolume` with the highest Priority (found in PhysicsVolume section of the volume's properties).

Zones

To provide a means of configuring gravity in large areas of your maps, you may use the 'escherZone' subclass of zoneInfo. This works exactly the same as a standard zoneInfo, except that it has a directional arrow for the direction of gravity in that zone.

The escherZone class also has an additional property, 'bNewMoveProjectiles' for setting whether the zone's gravity will effect certain projectiles. If true, projectiles will fall in the direction of the zone's gravity. If false, they will use default level gravity. The default is true. This does not effect projectiles that travel in a straight line (eg rockets). All the normal UT2004 weapons with falling projectiles have been subclassed for this (bio rifle, flak cannon, grenade launcher and mine layer), and so will work fine. If you wish your mod or mutator's projectiles to be compatible with Escher gravity, read up on the section about it at the bottom of this document. Weapons in Onslaught weapon lockers don't get replaced, and won't work. If you want these weapons in an onslaught map, stick them on an xWeaponBase instead.

You can find the escherZone class at **Actor/Info/ZoneInfo/escherZone**.

Volumes

To give you finer control over how gravity effects the player, use the `escherVolume` class. You create these volumes in the normal way, by right clicking on the volume creator with a brush selected. Simply select '`escherVolume`' as the volume type.

`escherVolumes` also have a '`bNewMoveProjectiles`' property, which works in exactly the same way as for `escherZones`. Projectiles will fall in the direction of the volume's gravity before that of the zones, and also respect volume priority as players do.

To setup the gravity for an `escherVolume`, you must add another actor and link it to the volume. Each volume may have **only one** of these actors. If more or less are found, log warnings will be generated for you, giving the name of the volume in error. Volumes may share a single actor, however.
You have three options:

Field gravity

This is where you specify a direction for the player to fall when in this volume. To do this, add a '`gravityDirection`' actor inside or near the volume (found under **`actor/gravityDirection`**). The `gravityDirection` actor has a directional arrow like the `escherZones`, which specifies the direction of the gravity for the volume.

Point Gravity

To create a volume where players will fall towards a point like a singularity, use the '`gravityPoint`' actor (found under **`actor/gravityDirection/gravityPoint`**). Players will automatically orient themselves on the roll axis so that their feet point down towards the `gravityPoint` actor.
Don't use this for things like walking on all sides of a rectangular prism, as it will pull players to the middle of each face when they jump. For this kind of thing, it is much better to use a field gravity volume for each face.

Inverse Point Gravity

The same as point gravity, except the players fall away from the point you specify. For this one, use the '`antiGravityPoint`' actor, under **`actor/gravityDirection/antiGravityPoint`**.

To link these actors to a volume, simply set the **tag** property of both to be the same (you can find this in the **`events`** section of each object's properties). Be sure not to get the names wrong here, or bad stuff happens! The easiest way to check is to use some capitalisation the first time you type the name, then type it all lower case the second time. If a match was found, the second tag property will change case to match the first.

The game won't care if you have gravity actors floating around without volumes assigned to them, but if you have a volume without a gravity actor, or more than one gravity actor, it will get angry and drop log warnings in UT2004.log for you, listing the name of the problematic volume and number of detected gravity actors so that you may easily fix it. Also note that none of these actors need be inside their corresponding volume. I recommend you use the same field gravity actor (not point or inverse point, obviously) to configure all volumes with the same gravity direction, to make things easier for yourself and enable you to make fine adjustments to the gravity in similar volumes through the same actor.

Final Things to Note

Slope tolerance

If you're feeling advanced, there is an additional property in the escherMut you added to your map before. You'll find it under the '**escherMut**' group in its properties. You can use this to set the angle tolerance which will be registered as a floor in zones and volumes. The default is 45 – this means that your player will be able to walk on any surface, so long as there is less than 45° difference between it and the gravity in that zone. Unless you're confident that you're setting your gravityDirection actors to the exact right rotation however, you should probably set this 5° or so above what you need. If you set this value to something stupid, it will clamp your entry to a 0-90° range internally. Note that this setting effects the distance underneath the player that will be counted as still being on the ground, so don't set it higher than you need or you will find your players walking slightly above the ground after coming off slopes.

Debugging

The escherMut also has a property you can set for debugging. Set bDebug to true to display useful info on your HUD in real time – the current gravity vector, actor, volume and zone. If you notice that the player still thinks it is in a volume after it leaves, this problem is probably due to touching volumes with the same gravity actor, as explained under 'things to note'. Volumes with non-planar faces will fuck it right up. The easiest way I have found to fix this behaviour is to work out the problematic zone and delete it and its actor, and rebuild. Just make sure you turn debug mode off before releasing your map! It will slow the game down an awful lot over time.

Mutator naming

The escher mutator you add into your map MUST have the name **escherMut0**. This is the default name, but if you have added multiple ones and deleted the old etc, you will probably have problems (where by problems I mean that nothing will work). If this happens, delete all the escherMuts in the level, save it, then reopen unrealEd and add your escherMut in again.

Perpendicular angles

If you want to make levels where the player walks around on walls at perpendicular angles, you'll need to note that they will only be able to travel in one direction fluidly. This is simply because a player can only be in one volume at a time. To fix this, you should either add higher priority 45° volumes in every corner (tedious), or make the corners of your volumes wedge shaped. I find that if you make the volumes overlap, and make the higher priority volume angle into the corner at a 1:2 gradient, this is just about the right amount for players to walk around in one direction, and be able to jump to get out of the higher priority volume in the other direction.

PlayTesting!!
I strongly recommend you playtest things vigorously before you release a map. Things like a point gravity actor without anything between it and the players could easily result in the player falling to the middle of the volume and getting stuck there. Point gravity actors are more designed for things like walking on the outside of a sphere (you would place the point gravity actor inside the sphere). There's lots of other ways to get people stuck as well, believe me.

Naming your maps
Also, I'd like to impose a naming convention and request that any map you release based on this system have the prefix DM-MCE- (or CTF-MCE-, DDOM-MCE- etc, for **M**. **C**. **E**scher) so that all the maps that use it are kept together in people's maplists and people know they will be running a mutator with the maps.

Player Starts
The normal playerStarts will only let you place them on flat ground, or they will throw errors when you compile the map. To get around this, use the **MCEPlayerStart** which can be found under **`Actor/NavigationPoint/SmallNavigationPoint/PlayerStart`** in the actor class browser. You can place these anywhere without unrealEd freaking out on you.

Weapon and Pickup Bases
The normal weapon and pickup bases will place their pickup directly above the base, which is not really what you want at all. Use the **MCE** variants of the regular bases to override this behaviour – the pickup will be spawned at the 'proper' location. All these classes can be found under **xPickupBase**, and are named in the same way as their regular UT counterparts. Note that the pickup itself will still rotate as if on the ground (fixing that would require subclassing every known pickup, which I'm damned if I was going to do). Oh well, at least it will be in the right place.

Compatibility with 3rd Party Weapon Projectiles
Skip over this if you're not a coder and don't plan on playing with it…

In order for projectiles to be properly effected by zones and volumes when using the 'bNewMoveProjectiles' option, they must have some special variables set. These are unfortunately constant variables, which means that you must pretty much rebuild their entire class tree to make them work.

Projectiles are bGameRelevant, which means they can't be replaced by a Mutator. Weapons aren't, however, so youll need to start by creating a subclass of your weapon. You change its defaults, giving it new `fireModeClass`es and a new `pickupClass` which basically needs to be subclassed to provide a backwards reference to this new weapon class. The new `fireModeClass`es are there to reference your new projectile subclass. Give your projectile subclass the following defaultProperties:

```
bNetTemporary=false
bUpdateSimulatedPosition = true
bReplicateMovement = true
bOnlyDirtyReplication = false
TossZ=0
```

Now that you've set these up, the projectile will work properly in the gravity volumes. Doing this makes them take up a great deal more network bandwidth than they usually would, so only use it if you think it's really necessary.

A sample setup for the bioRifle is included in this archive so that you can see how it's done. Note that it is completely unnecessary to make new subclasses if your weapon's projectiles aren't effected by gravity. Even if they are, you can always opt for having them fall regularly anyway, it's up to you.

Stick all the classes you need to make your mod or mutator compatible in some archive and compile them. *Please* name the archive something like escherProjectilesMYMOD.u (MYMOD being the name of your thing) so that all these packages are kept together. Also note that any use of these new weapons/projectiles online will require that players add your new package to their serverPackages as well. If they don't, clients don't get the weapon or the projectiles. Weird, but what can you do.

Now that you've done the hard part, actually getting this new projectile into the game in place of your old one is pretty simple. Go to the escherMut you placed in your map, and open its properties. You will see an array called `projectiledWeaponClasses` and one called `projectiledWeaponReplacements`. Put the class of your old pickup in the first array, the class of your new pickup in the second. So long as they are at the same position in the arrays, everything will work fine ingame. If your mod uses a different defaultWeapon to the assault rifle, youll need to put its class name (or subclassed one if it uses falling projectiles) as the `NewDefaultWeaponName` property.

Legal shit

If you make a map or a projectile package with this mutator, I would really like to know about it so that I can keep a list of them all together on the download page and make it easy for people to find these maps. Send me an email at pospi@spadgos.com.
Oh, and remember to always give credit where it's due. A little "uses pospi's Escher Physics Mutator" in your map readme or description would be much appreciated (:

Have fun!
pospi

25-10-2005

http://pospi.spadgos.com