Guidelines for creating an MD : MR character

Author : Pospi (pospi@spadgos.com)
Revised: 21st September 2005

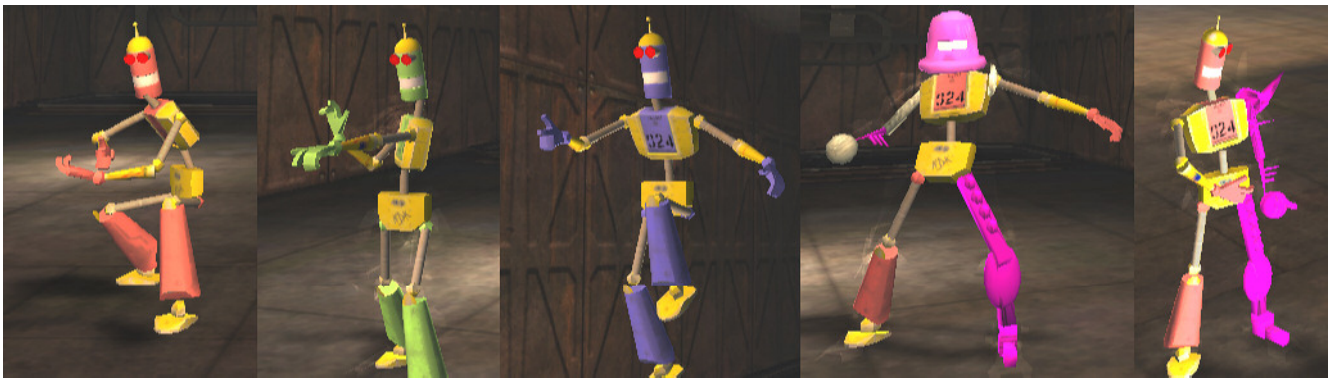
Contents

Explanation
Before you Begin
The Process
    Modelling
    Texturing
    Putting it all Together
The Character Designer
Distribution
Credits

Explanation

Welcome, artist type person!

This document is designed to show you all the necessary steps involved in making a custom character for *Marshmallow Duel : Mowbray's Revenge*. It includes a much cut-down version of the mod for you to test things ingame (in fact, this is probably based on code that's at least 4 months old), as well as source files for the character I will be going through so that you can learn by example.



MD:MR's animation system is different to that used normally in UT2004. Our characters are made up of static objects instead of a single mesh. Whilst this means our characters cannot blend between joints, it does mean they can swap different body parts in and out at will. The upshot of this is greater user customisability, since players have the power to create unique characters by combining pieces from many different models.

The material system our characters use is also unique, and allows players to select any colour from the rainbow as their character colour. As artists, you decide which parts of your character model to mask out for this custom colour, so you have complete control over it.

Creating a *MD:MR* character basically involves some modelling, texturing and text editing. There is no need to do any tedious rigging, weighting or animation for an *MD:MR* character. Yay.

A character will consist of 4 files – a texture package (UTX), staticMesh package (USX), script package (U) to point to the staticMeshes, and localisation file (INT) to point to the script package.

Whilst I won't go over the process in every conceivable detail, I will lay out the basic instructions for each of these steps so that you know what to do. Basically, this tutorial is for people who know how to model, skin and write some simple text files.

Remember when designing your character that it will probably be used alongside parts from other characters. Keep things modular, but don't be afraid to try your own thing.

Before you Begin

Included in this package is a file named 'mduelCharDesigner.zip'. This is a very, very cut-down version of the *MD:MR* mod, and exists only for you to use for compiling and testing characters. It has no features other than the character designer system, and will not work online or on a LAN. Bots will show up invisible, so don't go testing with bots either.

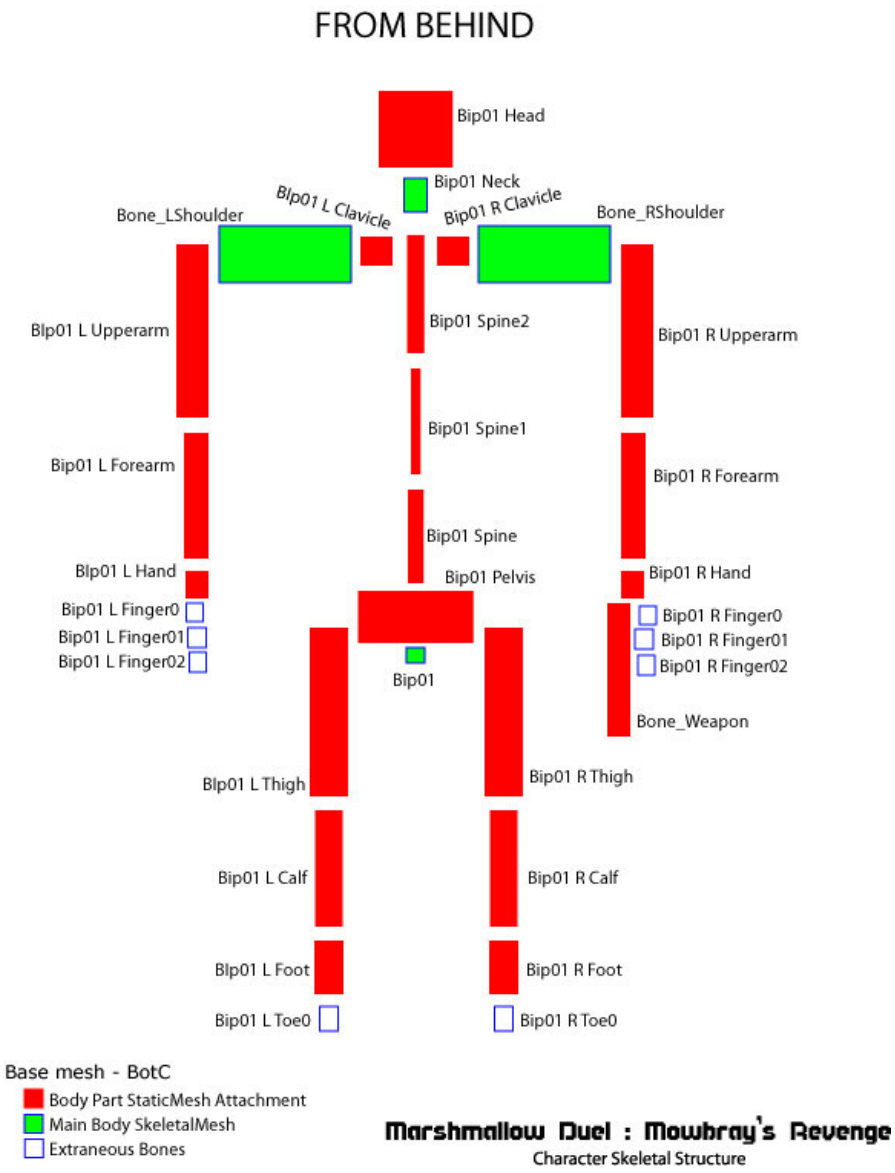
Basically, you can just extract this file to your UT2004 directory. It will create a new folder called 'mduelCharDesigner' that contains all the data needed for you to test and manage your character creations. When the full *MD:MR* is released, your characters can be copied over and will work perfectly.

More on using this system is discussed towards the end of this document. For now, you should just know that it exists. The tutorial assumes you have extracted it to your UT2004 folder.

Modelling

This part of the tutorial is aimed at users of Lightwave, since Lightwave is the 3D program most used for staticMesh creation in UT2004 (and also because it's my program of choice and the only 3D package I know anything about). However, the instructions given are very general, and can be applied to Maya, 3DS Max or whatever other bollocks you like to use (just kidding).

To create an *MD:MR* character, you will need to create a static mesh for each element of their limbs. These meshes are then imported into UnrealEd as staticMesh objects that can be used in the game. So, here's the player's skeleton with its bones colour coded:



The red bones are the only ones you need to worry about. Each of these (except for the weapon bone) needs a staticMesh attached to it. You can only attach stuff to the red bones, so don't worry about anything else.

Usually, each selectable limb will choose a single staticMesh that attaches to a single bone (confusing sentence, yes?). However, there are some special cases:

- Ingame, players have a single selector for their torso. The torso is actually comprised of up to 4 staticMeshes, which attach to the pelvis, spine,

spine1 and spine2 bones respectively. You do not have to use all of these attachments, however the pelvis is **compulsory** if you are creating a torso (your torso won't show up if you don't have one). Feel free to use or ignore the other 3 bones as you see fit.
- The shoulder meshes (the ones that attach to the clavicle bones) are more for shoulder pad sort of stuff and these are optional for players to choose.

All these attachments are optional for you as an artist when you create a character. You are quite free to create as full or as limited a character set as you wish. For example, say you don't like the way a particular guy's head looks – well, create your own head and make a new character that is just a head. Your new head choice will be added to the game so that players can access it as well.

Also note your ability to reuse objects. For example, say you are creating an almost perfectly symmetrical character with simple cylindrical objects for thighs. You could reuse this thigh object for both thighs, since it won't change too much from being attached to different bones. There are exceptions to this however, but they are explained below where appropriate.

The main difficulty you are going to have when you do your modelling for *MD:MR* is the scale and orientation of your actual meshes. Each piece needs to be in the right position and rotation to attach to its bone properly and look convincing.

The skeleton that everything will be attaching to is, in fact, the default skeleton for the bot characters that come with UT2004. I've included the bot skeleton in this package in a few formats in case you want to use it as a reference for model sizes, etc. Just make sure you size it to the correct scale so that your conversions into unrealEd work out properly. More on that in the next few paragraphs.

Just to note, the bot skeleton will be animated differently when *MD:MR* is finished, so things will move differently and with more awesome. You don't need to worry about any of that though, so let's move on.

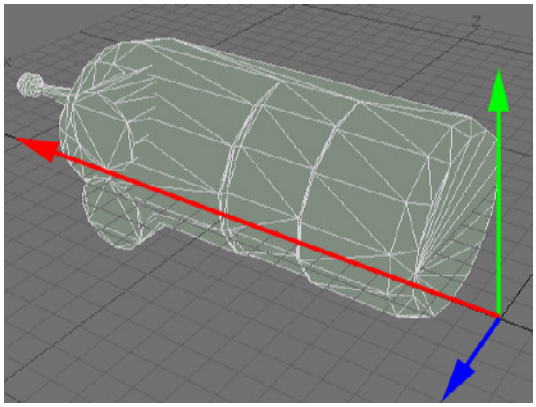
To begin, let's explain some things.

- Lightwave uses a Y-up coordinate system. Unreal uses a Z-up coordinate system. This means you have to rotate your meshes all weird in Lightwave to get them to import into UnrealEd properly. The same is probably true of other packages, but there's always the possibility that your conversion plugin will rotate them for you, so I'm not at all sure. The best bet here is to do everything the exact same way as I describe it, then if things turn out strangely later, go back and apply rotations to fix this for all your objects. They'll be simple 90° rotations anyway, and you can guesstimate them just by looking.
- Unreal uses its own coordinate system, which will be different from what your 3D program uses. So your meshes must be modelled at the correct scale. For example, I thought it would be easy to use the bot rig as a guide when I did my modelling. So I converted it from a 3ds max file into a Lightwave object. However the scale was then all wrong. And its entirely possible that whatever you use to get animated meshes into Unreal uses a different scale than your staticMesh importer (I doubt it though). So, as a guide, a character in UT2004 is **approximately 96 unreal units high**. I found that when I scaled the bot mesh to that size things worked out pretty nicely. You can always use this approach to template sizes if you wish. Anyway, the rest of this tutorial uses Unreal Units as a measurement. Conveniently, **in Lightwave 1m = 1UU**.

For the next part, I'm going to assume you know what youre doing when it comes to modelling. If not, there's some great tutorials on the UnrealWiki and UDN. I've thought about the best way to show the orientations and scales of things, since it's really all you need to know, and I think the best way is screenshots of the objects that make up the Robot test character. I recommend you keep your originals in the orientation you are most comfortable with working on them and then save separate files of the rotated ones. But hey, it's up to you.

Each of the shots has 3 lines drawn on it to show the direction of the axes. They point from negative to positive. Their colours are:
X – red
Y – green
Z – blue
These lines also radiate from the world origin (point at 0,0,0), since this will convert across to the origin of the staticMeshes when they are ingame (basically, the starting point of each bone your models are attached to).

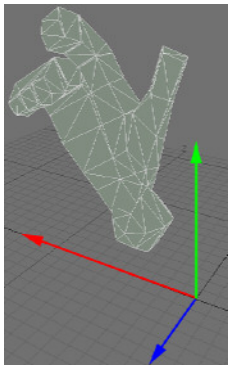
Head



The head should usually be positioned at the world origin, but because of the robot's long neck, his is positioned higher. This means that with other models his head will float a little off his body, but these sorts of small discrepancies will not matter too much in the end and really only add to the game's idea of characters that are 'loosely cobbled together'.

The head should be about **12-16UU** high.

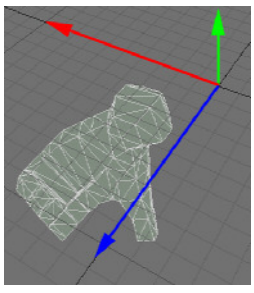
Left Shoulder



These shoulder guards look like hands because the robot didn't come with shoulder pads and I couldn't be bothered making any.

In this image, the fingers curl back towards his back and the thumb points inward, towards his ear. The model is positioned above the origin so that the shoulder guards seem to join to his chest instead of coming from inside it. It is also moved slightly along the X axis so that the shoulder guards do not interfere with his head. Basically, remember that the bones the shoulder guards attach to are deep inside your characters neck.

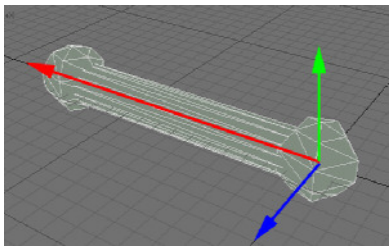
Right Shoulder



The right shoulder is much the same as the left, however it is mirrored in the Y axis. If you were to think about the way the clavicle bones face, this must make sense. I just guessed though.

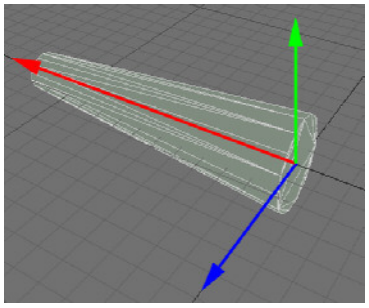
Both shoulders should be about **8UU** high, including their offset from the origin.

Upper Arms



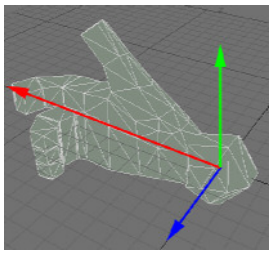
The upper arms are fairly straightforward. The robot uses the same model for left and right upper arm, though for asymmetrical characters one of your arms might come out rotated 180° around the X axis once it is ingame. This applies for most limb appendages, so it's a good rule of thumb if one of your limbs is rotated lengthwise.
The upper arms should be about **15-17UU** long.

Lower Arms

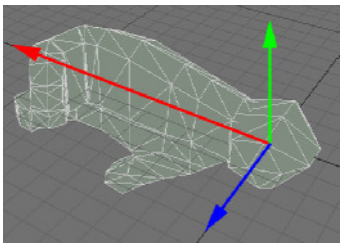


Lower arms are pretty straightforward again. Remember everything I said about the upper arm, but for lower arms their length should be about **13-15UU**.

Hands



Left



Right
The hands work about the same as the shoulder pieces. For your left, the thumb points up and fingers point along Z. For the right, the hand is mirrored on Y.

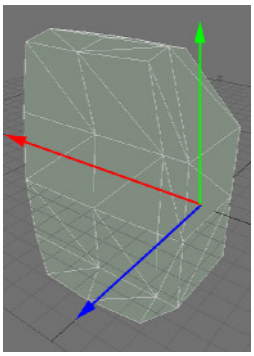
The right hand should be modelled as if it is holding something, so that weapons can be attached to the weapon bone and the character will look like it is actually holding it. Do what you like with the left.
Hands should be about **7-10UU** long.

Torso

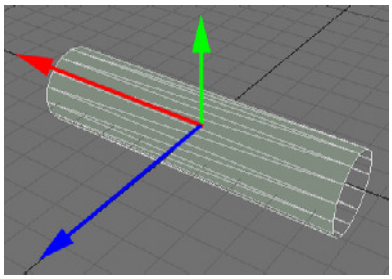
The torso is certainly the most complicated, being that it is composed of many different pieces. We'll go through them on the next page.

Torso Parts

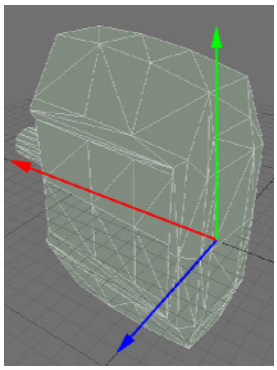
Pelvis



Spine Sections



Chest



All the torso sections face forwards along the Z axis. So in these images, you can see the front of the model.

Pelvis

The pelvis is the lower piece, which will attach to the character's pelvic bone. It's important to proportion the pelvis correctly so that the legs stick to it properly. Remember that the pelvis is the only mandatory part of your character's torso (though you would probably at least have a chest piece as well :P)
The pelvis is about **8-11UU** long.

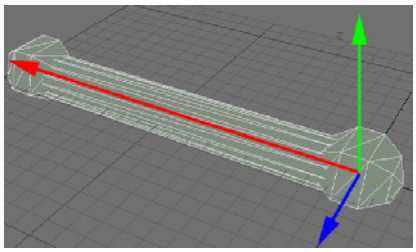
Spine Sections

The spine sections are the middle sections of your character's torso. You are given the option of having 2 separate pieces, but to be honest, unless you really want to fake smooth bending you're only ever going to need to use the 'spine1' bone. The 'spine' bone is much lower down (almost as low as the torso), just so you know. The 'spine1' bone is about half way between the pelvis and chest.
If you're going to use both attachments, then they should start at the world origin and go along the X axis in a similar way to the arm pieces. In this case, a good length for each is about **3 or 4UU**, but the lower piece should probably be a bit smaller.
If you prefer the easier option of using the higher 'spine1' attachment, just create your middle torso piece as in the diagram above, roughly centered on the world origin. The top (+X axis) section should be a little longer than the bottom.
A good length for this second approach is **7-9UU**.

Chest

There's not much to say about the chest. It's basically the same deal as the pelvis. Just make sure it's wide enough so that the arms hook into it in a realistic way. Shoulder guards are also a concern, but they're supposed to look floaty anyway so it isn't that important. Make sure you don't make your chest too tall, or you will have trouble with heads clipping through its neck (this is part of why the robot model's head floats away from other bodies but sticks nicely to robot).
A chest should probably be somewhere between **13 and 16UU**.

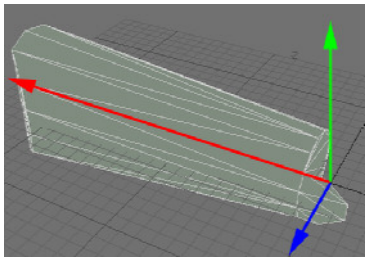
Thighs



Nothing too fancy to see here. The thigh and shin pieces are about the same sort of thing as the upper and lower arms.

A good thigh is about **20-25UU**.

Shins

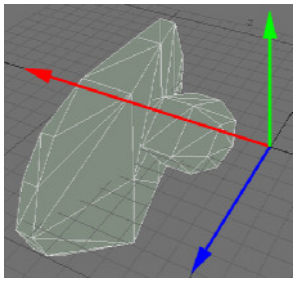


Once again, nothing too special. Shins can be a bit longer than the bone they are attached to, depending on the way you want to do your foot. For most cases, I recommend you make the shin the correct length and have your feet rotating about their origin. The feet in this case should be modelled as boots – ie they should take some of the ankle into account. The best test for feet is to look at your character when standing still and make sure he is actually touching the ground :P

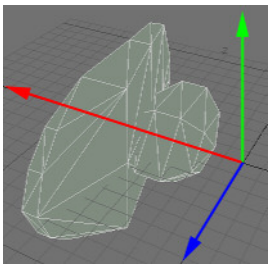
The robot does his legs the other way – his shins are a little longer, but his feet are more flat and have no ankles. This means they are moved away from their origin, as you will see next. Either way is fine, really, but combinations of these two types of models might look silly (that's good!).
A good shin should be **18-23UU**.

Feet

Right



Left



The feet are again, the same sort of thing as the hands and shoulder guards. Like I said in the shins part, the robot feet are moved a bit away from the origin, when most feet would probably have an ankle-like boot heading up there.

On the right foot, the big toe (if the robot had one) is on the underneath side. On the left, the big toe is on the top.
A good foot is about **7UU from origin to base**, and **10-15UU from toe to ankle**.

Texturing

Texturing is probably the easy part. I'm going to assume you've already UV'ed your model and are ready to start texturing in whichever painting program you prefer. For me, it's Photoshop CS2, but that's really a moot point.

The way you texture your models is entirely up to you. Use as many UVs per model as you wish. Use many different textures, or use a single texture map for all your models. It doesn't matter, so long as you design your textures ingame in the way outlined below. Just be mindful of the resources you'd be using if you went and textured every limb with a 1024x1024 texture :P

The robot model, being somebody else's work, came with textures for me so that saves a bit of time writing this tutorial. He uses the one texture/UV for all the different body parts, since he started off as a normal UT2004 character. Here's the initial texture:

Now, I chose to make all the blue parts of the texture of configurable colour. You don't necessarily need to have a configurable area on every different limb, but remember that the whole point is to have enough custom colour visible that a players' colour is apparent to others.

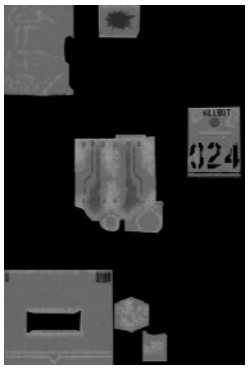
So, to begin, the simplest thing to do is mask out the areas youll be changing. In Photoshop, I like to use the lasso tool, or just make a new layer mask and paint on it if finer detail is required.

The next step is to desaturate all the areas you want to change. If you've started your character from scratch, youll probably just make those areas in grey and white to begin with. But, I'll talk about it anyway just in case. You must do this for every texture you want to have custom colouring on.

Here's the texture with the chosen areas desaturated. In Photoshop, you can do this to the selected area just by going Image>Adjustments>Desaturate. In other packages, you might have to copy to a greyscale image and paste back. Up to your chosen method anyway.

This will be the main diffuse texture your model uses (or one of them if you use multiples). Save as a normal 24-bit bitmap.

Now, copy the desaturated areas to a new image, and fill in black behind them. This will be your masking image for the custom colours to appear on. You might want to up the contrast of this one so that its very hard black and white, or the skin might look washed out later.



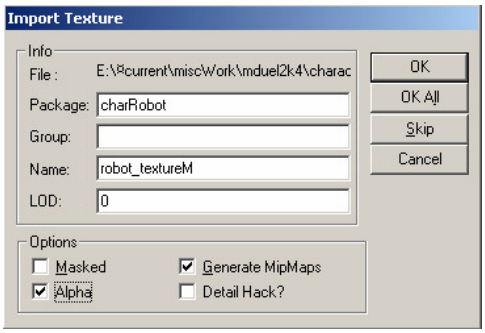
Save this file as a 32-bit targa image with the alpha channel set to the same stuff as the black and white image you've already created. (You can do this in Photoshop by merging layers, copying everything, making a new channel in the channels pallete and pasting it in again).

Putting it all Together

Ok, so now you should have a bunch of models and textures ready to import into UnrealEd. You should start now by importing your textures, so that when you import your staticMeshes, they can find everything they need to display properly.

When importing our textures you should setup the materials system properly to take advantage of the colouring system. If you are comfortable with UnrealEd, then you might want to go ahead. Basically, the skin is built as a Shader with Combiner as its diffuse component. This combiner is made up of your diffuse texture as material1, masking texture as mask, and material2 left blank (this is important). Note that this step isn't required if you don't wish to use configurable colours on your model. If you do (and you should), read on.

Fire up UnrealEd. Go to the texture browser. Choose **File>Import** and import the textures you will need for your model. Give them a nicely named, **unique** texture package that implies they are for your model (if you haven't thought of a name for your character now might be a good time).

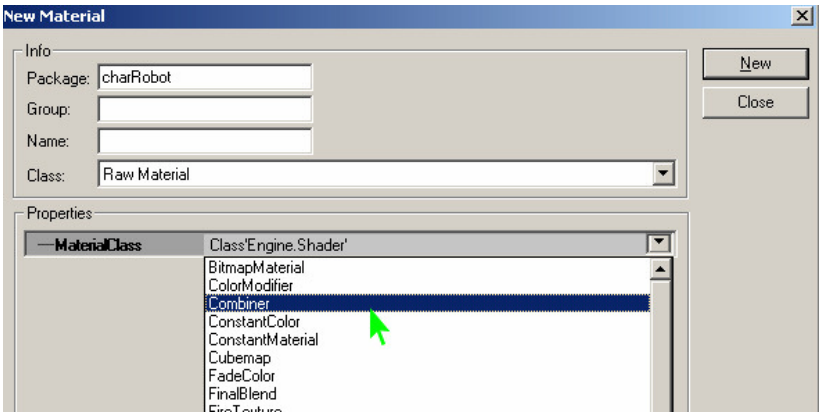


For the masking texture, make sure the 'Alpha' box is checked. You should also generate mipmaps for both. If your diffuse texture has transparency in it (I don't know, maybe you are making feathers or hair or something), you'll want alpha for that as well.

Feel free to use a grouping if you want – it's only for organisation and just means more typing for you later on. If you're making more than one character you can of course put everything in a single package (both for textures and staticMeshes), so in such a case grouping might be handy.

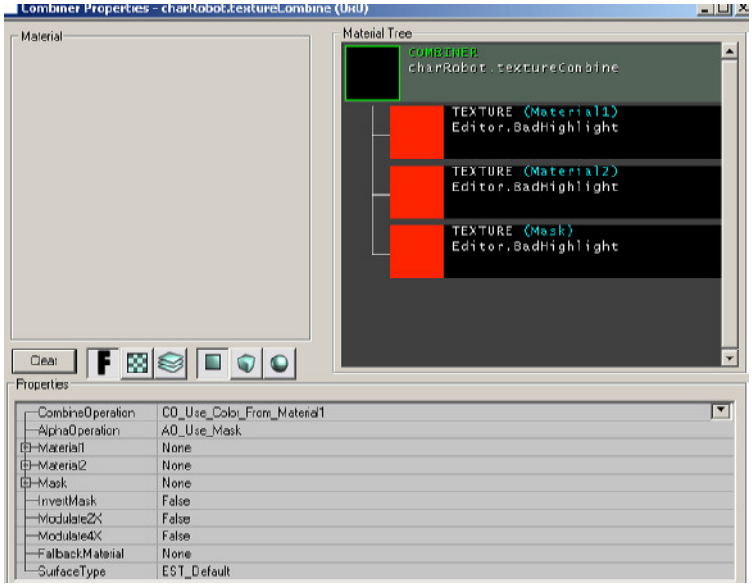
Ok, so now both your textures are in UnrealEd. They're taking up a lot of space though, so let's fix that. Right click on one and select 'Compress'. You will see DXT1, DXT3 and DXT5 options. DXT1 is best for textures with no transparency, and gives the best compression. DXT3 is for textures with simple alpha (think GIF files) and gives medium compression. DXT5 gives the lowest compression, but features full alpha. You'll probably want DXT1 for your main texture and DXT5 for your mask. But if you've done different things just choose what fits best.

Now your textures are taking up a fraction of their former space, and people will have less trouble downloading them. Next step is to build your shader.



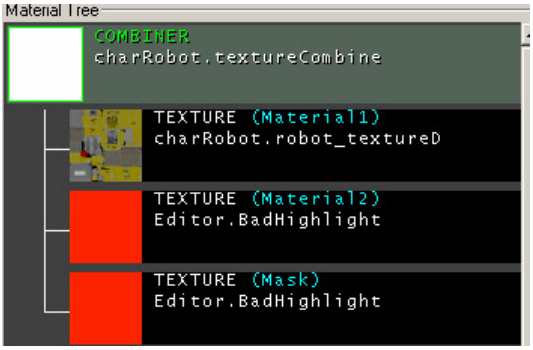
Choose **File>New**. Choose **'Engine.Combiner'** from the list. Give it a good name and make sure it's going to be created in your current package.

The following screen will pop up:

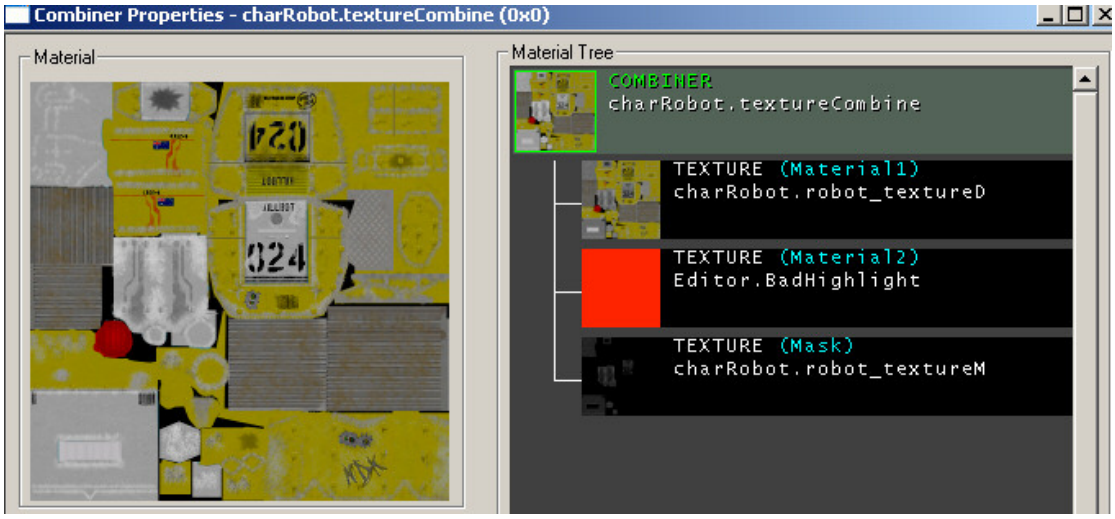


Set the following:

CombineOperation = CO_AlphaBlend_With_Mask
AlphaOperation = AO_Use_Mask



Now click on the 'None' word next to 'Material1'. Move the combiner properties out of the way and find your normal diffuse texture in the texture browser. Select it (when selected, a grey rectangle should be behind it). Now go back to the combiner properties and press 'Use'. Your combiner's material tree (the graph like thing at the top right) should now look like this ←

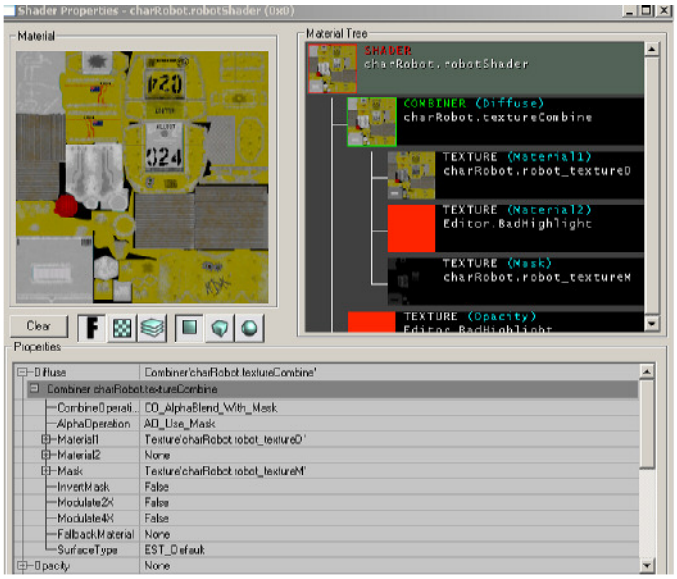


Do the same for the 'Mask' property, this time selecting your masking texture. You should be able to see something in the material preview window now.

That's it for our combiner. **Make sure** its material2 property is left blank – this is how the shader system knows to recognise it as a compatible texture (why would anyone else use a combiner without a second material to combine?).

Go back to the texture browser. Your combiner should be in there now, designated with a green border around it. It should basically look the same as your base texture, with slightly brighter areas where you have masked.

Now, go to **File>New** again, and this time select **'Engine.Shader'**. Its properties window will now come up. Set its 'Diffuse' property to the combiner you have just created, in the same way as when you set combiner properties up before. You'll now have something like this:



You're done. That's everything you need to take advantage of the *MD:MR* materials system. You can save your texture package now – by default it will go into the **UT2004/Textures** folder. For the character testing system though, you'll need it to be in **UT2004/mduelCharDesigner/Textures**. Save it there now or move it later, it doesn't really matter.

If you're feeling advanced, you can go nuts and add in other things to your shader like specular, self illumination etc. That's up to you though.

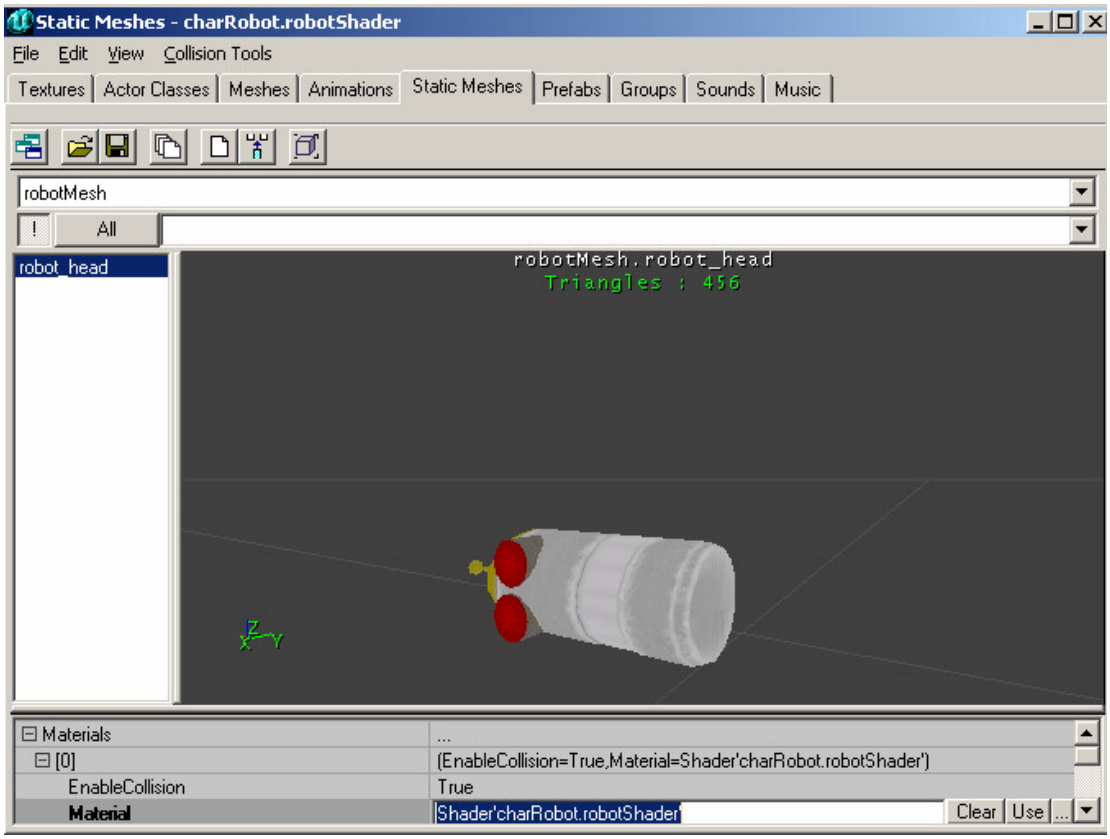
On to the staticMeshes!

Importing these is much less complicated than the textures. If you've named everything correctly, you should be able to simply import them and everything will link up. Your texture package must be loaded however, or else they won't be able to find the right textures.

For Lightwave, simply go to the staticMesh browser and select **File>Import**, then point to your LWO files. For other 3D programs, follow their respective instructions.

Make sure your staticMesh has its materials set to the shader you designed above. You could do this automatically by naming your surfaces in Lightwave so that they match up with the correct texture in UnrealEd, but for me I prefer the security of doing it myself anyway. You can select a staticMesh's materials the

same way as you selected textures for the combiner above. Just select your shader in the texture browser, then go back to the staticMesh browser and hit the **'use'** button.

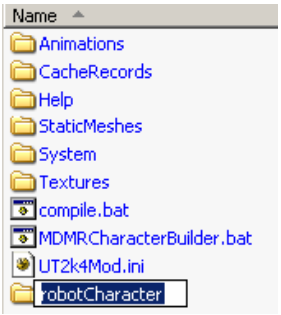


To see the texture applied to your staticMesh in real time, you should press the **'!'** button at the top left of the staticMesh view window. You will probably also need to rotate and move the camera around to see your mesh, since it's going to be really small in comparison to the staticMeshes used for level geometry.

If your staticMesh uses more than once surface, just expand the **'Materials'** property at the bottom of the window and choose the relevant shader for each piece.

When you've done all the relevant meshes you need, just save the package in the proper place (should be **UT2004/mduelCharDesigner/StaticMeshes**).

Now you have all your assets ingame and ready to use. The next steps involve telling the game where your meshes are located and how to use them. Once you've done this, you needn't do it again a second time around. Let's say you find out your head is too small. Well, just go back and change it, then import it over the top of your old head in the staticMesh browser. The game will just use the new head in place of the old one.



Ok, so to begin, you must make a new folder under the **'mduelCharDesigner'** folder in your UT2004 directory. This will be the name of the *.U package you end up creating for your model, so give it a meaningful name.

Under this folder, create another folder named 'classes'. This is where you will make a simple text file to load all your model's staticMesh pieces.

Now create a new text file in your classes folder. Rename it to whatever you like (but NO SPACES! And no starting with a number!), and give it the extension *.UC instead of *.TXT (if you're hiding file extensions I think you'll need to show them in order to do that).

Open up said text file in whatever text editor you're most comfortable with. I personally prefer UltraEdit, but I'll do this exercise in notepad (shudder) since everyone has it.

Copy and paste the following text in:

```
class WHATEVER extends duelerInfo;

defaultProperties
{
	attachmentActors(0) =StaticMesh'' //head
	attachmentActors(1) =StaticMesh'' //rshoulder
	attachmentActors(2) =StaticMesh'' //lshoulder
	attachmentActors(3) =StaticMesh'' //rupperarm
	attachmentActors(4) =StaticMesh'' //lupperarm
	attachmentActors(5) =StaticMesh'' //rlowerarm
	attachmentActors(6) =StaticMesh'' //llowerarm
	attachmentActors(7) =StaticMesh'' //rhand
	attachmentActors(8) =StaticMesh'' //lhand
	attachmentActors(9) =StaticMesh'' //rthigh
	attachmentActors(10)=StaticMesh'' //lthigh
	attachmentActors(11)=StaticMesh'' //rcalf
	attachmentActors(12)=StaticMesh'' //lcalf
	attachmentActors(13)=StaticMesh'' //rfoot
	attachmentActors(14)=StaticMesh'' //lfoot
	attachmentActors(15)=StaticMesh'' //hips [MANDATORY FOR CHEST!]
	attachmentActors(18)=StaticMesh'' //lower spine
	attachmentActors(16)=StaticMesh'' //upper spine
	attachmentActors(17)=StaticMesh'' //chest

	setName="My Characters Name"
}
```

Ok. Replace 'WHATEVER' with the name of your new file (without the .UC at the end). Now you simply have to add in the names of your staticMeshes that make up the character's body. The notation used is **Package.Group.Mesh**. So, for the robot character, his head is defined like so:

```
attachmentActors(0) =StaticMesh'charRobot.robot_head' //head
```

Notice there is no group part since I did not use any groups when I imported. If you can't remember the names of things, you can just open up your staticMesh package again in UnrealEd and select the meshes - it will give you the full path to them at the top of the window.

This system is perfectly open to you. If you don't wish to specify a mesh for a particular limb, just delete the relevant line. For example, the robot has no lower spine attachment, so I delete the line
`attachmentActors(18)=StaticMesh'' //lower spine`.

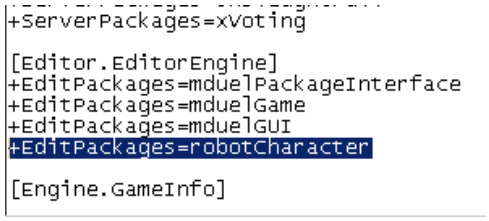
Remember, you can also reuse the same mesh for different parts. To do so, just use the same path multiple times. Another example from robot:

```
attachmentActors(3) =StaticMesh'charRobot.robot_armUpper'   //rupperarm
attachmentActors(4) =StaticMesh'charRobot.robot_armUpper'   //lupperarm
```

So now you have set up a reference to your meshes. Also remember to change the last line (`setName="My Characters Name"`) so that your character has a unique name. This is the name that will show up in the menus when players select body parts from your character. Make sure you keep the double quotes in!

You must now compile this *.UC file into a *.U file.

Go into the **'System'** folder inside mduelCharDesigner. If the file **'mduelCharDesigner.ini'** exists, delete it. The game will make a new one in a minute anyway. Open up **'default.ini'** in notepad or another text editor.



Add a line to the [Editor.EditorEngine] section as shown. The 'robotCharacter' part shown here should be the name of the folder you first created to put your 'classes' folder and *.UC file in.

Save the file and close it.

Now you are ready to compile your *.U file. Note again that if you are creating multiple characters, you can have more than one *.UC file inside your folder to compile (one for each character).

Anyway, simply go to the root of your **'mduelCharDesigner'** folder and run the **compile.bat** file located there. If you did everything properly, it should spit out a heap of junk and then say '0 errors, 0 warnings'. If not, it will tell you the line you messed up on so you can go back and fix it. Press enter to close the window it made.

Now you should have a *.U file in your mduelCharDesigner/system folder. The last step is to tell the game to look in this file.

Create another new text file, this time in your **mduelCharDesigner/system** folder. Rename it like you did for the *.UC file, but give it the extension *.INT. Open it up in your text editor and paste the following in:

```
[Public]
Object=(Name=PACKAGE.CLASS,Class=Class,MetaClass=mduelGame.duelerInfo)
```

You now need to change 'PACKAGE' to the name of the folder you created before (for me it is 'robotCharacter') and 'CLASS' to the name of the *.UC file you created. So the robot character has something along the lines of this:

```
Object=(Name=robotCharacter.DUELERRobot,Class=Class,MetaClass=mduelGame.duelerInfo)
```

If you’ve got more than one character, again, just add in more ‘Object=’ lines, one to reference the *.UC file of each character.

Save your *.INT file. That’s it! Everything should show up ingame now!

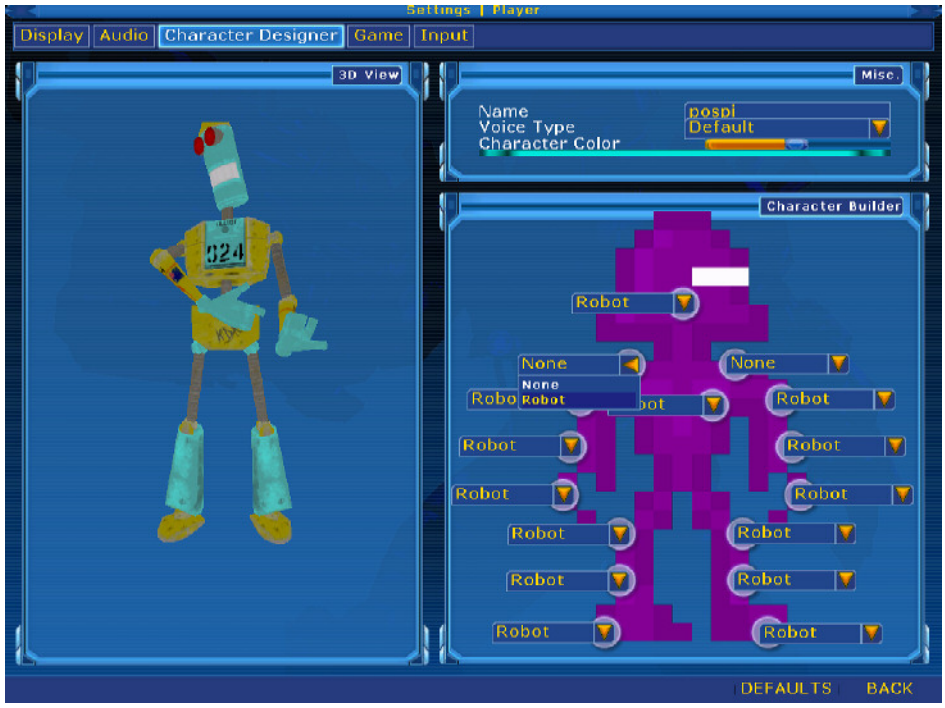
The Character Designer

So you've gotten this far. Everything you need to do to create an *MD:MR* character is finished. But how to you see what it looks like ingame?

Go to the **'mduelCharDesigner'** folder in your UT2004 directory. Run the **'MDMRCharacterBuilder.bat'** file. This will load up UT pretty normally.

On the main menu screen, click on 'Settings' to take you to the settings page:

It should be pretty obvious what to click on next. The character designer brings up this screen:



You use the 'character color' slider at the top to change the custom colour of your character. The selection boxes laid out below let you choose the different body parts for testing. The robot character is included by default, so you can mix and match him with your creations to see how things work together.

Of course this isn't much of a way to see how your character looks, so when you're done, just go back to the main menu. Your choices will be saved.

Now, head to an instant action game (other things might work a bit, but probably not very well at all since that code isn't done yet). Choose **'Simple UT DM Test'** as your gametype (nothing will happen if you don't do this). Take out all the bots (they'll just get in the way) and start a new game.

You should see the character choices you made before reflected in the guy standing in front of your camera. Note that there is a small bug at the moment that I was too lazy to fix – your character will load shoulder guards the first time you play even if you have specified 'none'. To correct this, go to the settings menu, switch to some guards, go back to the game, then go back to settings and switch to none again. They should be gone now :P

You can change your character's appearance as you play, without disrupting gameplay. You can also use the following keys to get a better look at him (and take cool screenshots if you really want to).

I : Disable free camera
O : Enable free camera
P : Show/Hide hud

And that's the end! If some things look a bit iffy, just go back and reimport them over the top of the old assets. That way, you don't have to update anything else.

If you create a custom character and would like it featured in our mod when it is released, you can either email pospi with details using the email address at the top of this document, or post it at the Marshmallow Duel : Mowbray's Revenge forums.

Read on to see the necessary steps involved in distributing your character.

Distribution

There are essentially three groups of files you need to distribute to people in order for them to be able to use your models – Textures, StaticMeshes and System files.

Below is a simple directory structure showing you which files you must include in your character download.

UT2004

- **StaticMeshes**
  - Any *.usx files with your models inside
- **System**
  - Any *.int and *.u files your characters use
- **Textures**
  - Any *.utx files with your textures inside

Credits

'BotC' character model source file © Epic Games

'Robot' model created by Michael Lane -
http://student.ci.qut.edu.au/~n4416295/folio_main.htm

Marshmallow Duel : Mowbray's Revenge is based on the *Marshmallow Duel* game originally created by Duncan Gill in 1996.
Please visit www.marshmallowduel.com for more about this classic game!