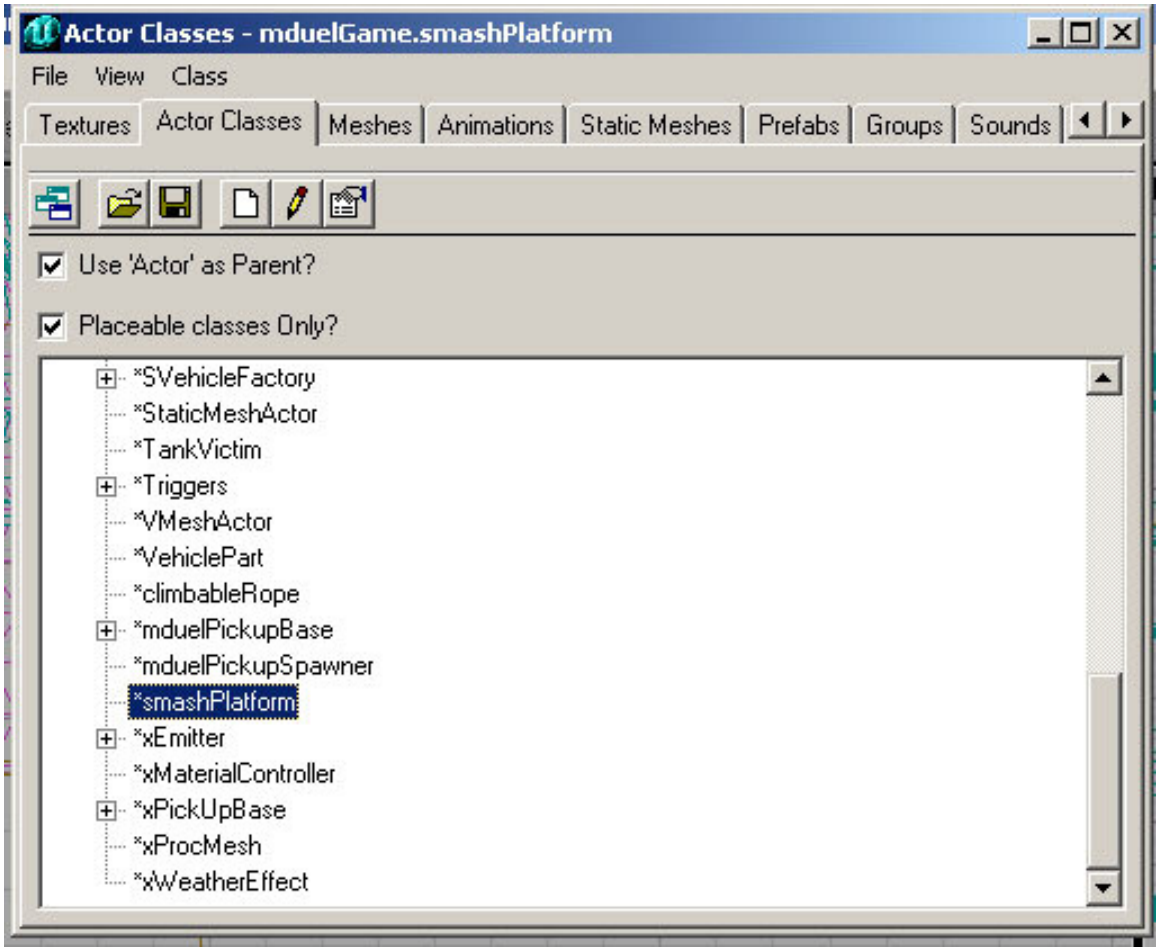
This is a mapping tutorial for the UT2k4 mod Marshmallow Duel: Mowbray's revenge. It assumes you have experience with unrealEd already.

There are several main parts that make up a typical mduel map which set it apart from a normal UT map. These are smash platforms, pickup spawners and their pickups, climbable ropes, and the all important deadly marshmallow at the bottom of the map. All of these elements are optional, but most of these are pretty important for it to really be a Marshmallow Duel level.

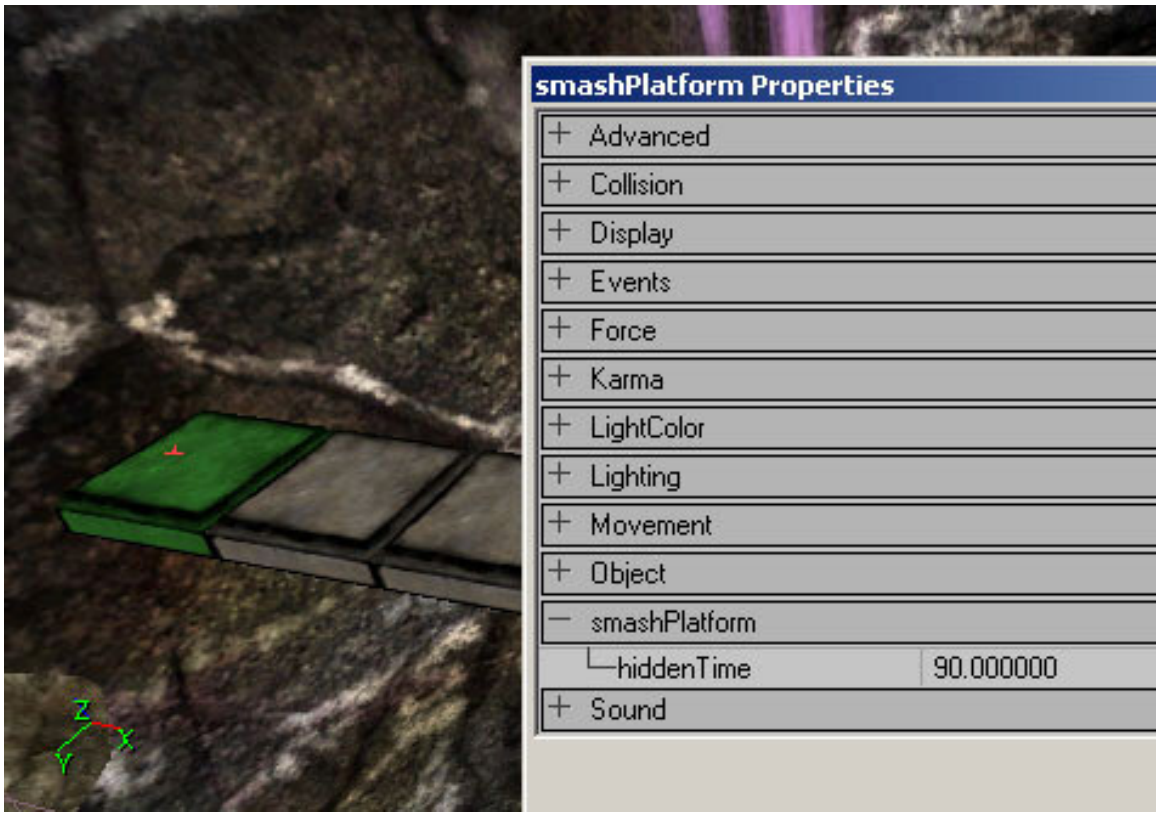
SMASH PLATFORMS

The platforms and main playing areas are usually made up mostly of Smash Platforms. A Smash Platform is a new actor which is basically a static mesh with some added logic that allows it to be destroyed by grenades and tnt, and respawn after a certain period of time. Although the game options also allow for setting smash platforms to not respawn, which you should keep in mind when building levels. Of course, you can still build playing areas with BSP, regular static meshes and terrain to add variety, however some if not most of the action should take place on these destroyable meshes above the Marshmallow.



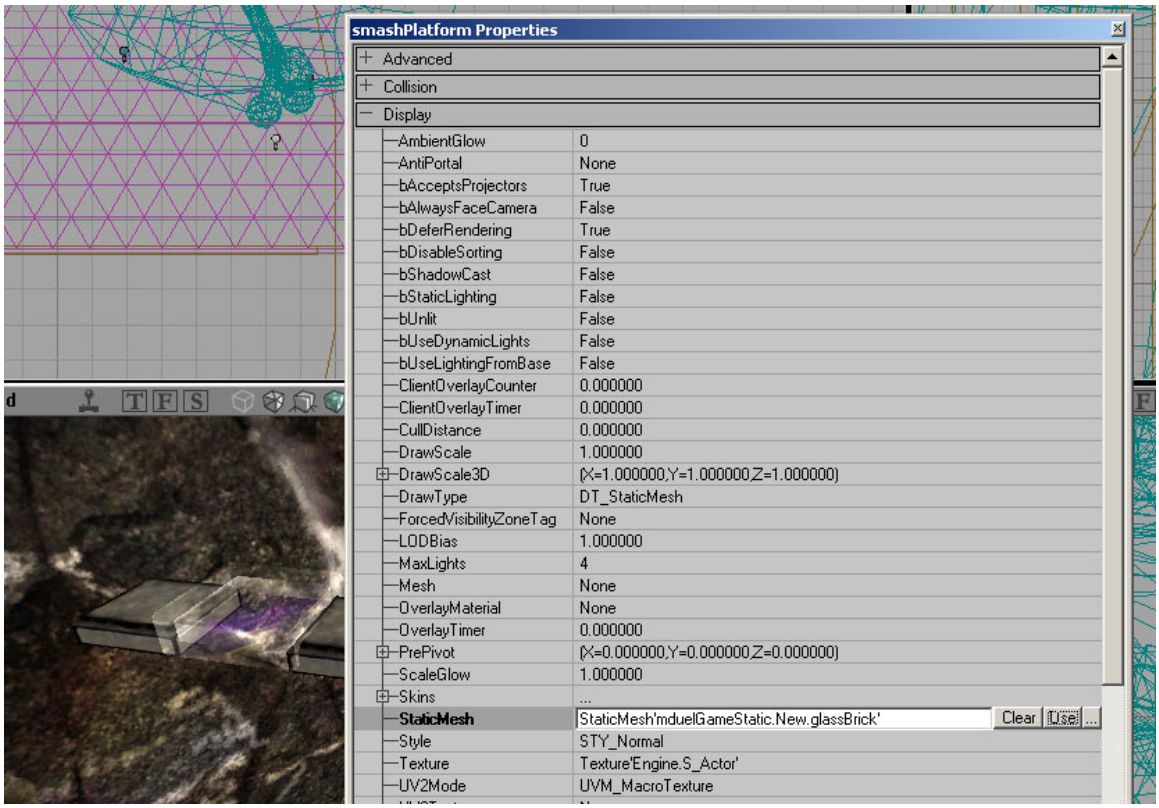
To place a smash platform, go to the Actor browser and find the smashPlatform actor. It's usually near the bottom of the list. At this points its probably worth noting that you need to open unrealed with the 'runUnrealEd.bat' file, so that unrealed is opened with the mduel settings and actors loaded. Or you can just type it in yourself i think its just "unrealEd.exe -mod=mduel".

Anyway if you select the actor, then right click where you want to place it as usual, and place the smashPlatform Actor. You should get the default Marshmallow Duel grey square platform mesh. From here you can just start placing your platforms, or you have a few options; you can play with the smashPlatform settings or change the static mesh used.



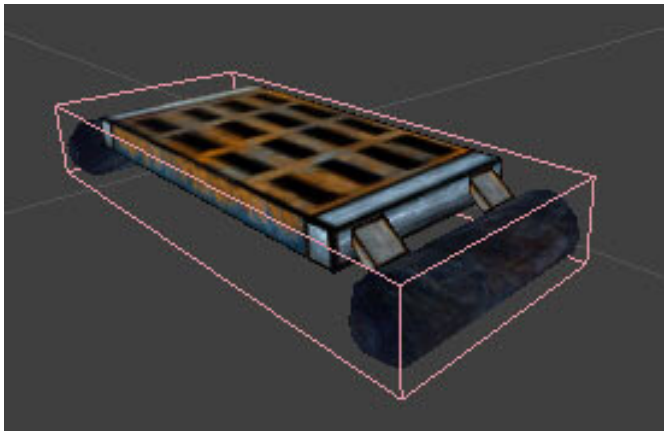
The only smashPlatform setting at the moment is 'hiddenTime' which gives you an option to change the time, in seconds, it takes for the smashPlatform to reappear after it has been smashed. This is under smashPlatform properties, and go to the smashPlatform submenu. The default is 2 minutes.

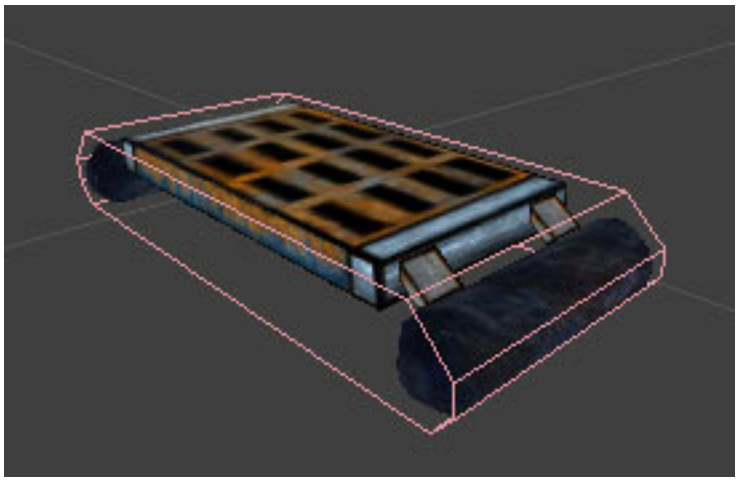
To change the static mesh, just go to the Actor's properties and to the 'Display' submenu. In here there is a parameter with the name of the mesh that is being used, you can just select from the static mesh browser, go back to the properties and click used, your smashPlatform should change straight away to the new mesh. If you plan on using the meshes that come with Marshmallow Duel for your level, the package is 'mduelGameStatic', there are a few variations on the classic grey platform available in the 'Classic' group, as well as some catwalk and glass brick meshes in the 'New' group.



There are also some tree trunks, branches and mallow blob meshes which have been used to make destructable marshmallow trees in a few of the maps that come with the game. These are good for adding some interesting variation, which is able to be destroyed piece by piece, although it can be a bit awkward for the player to use as platforms.

Which brings me to collisions. If you are importing your own static mesh, you should be careful about the collision settings. Obviously you can use the default colisions of the static mesh itself, although this is not very good for performance and could slow down your map. Ideally you want a flat surface on the top surface of the collision box, where the player is going to walk. If you have curved surfaces at the edges this can interfere with the player or pickup physics, especially the player's rolling. Even though you might use some static meshes with more complicated surfaces, it is best to use flat collision box surfaces when possible.


Good collision



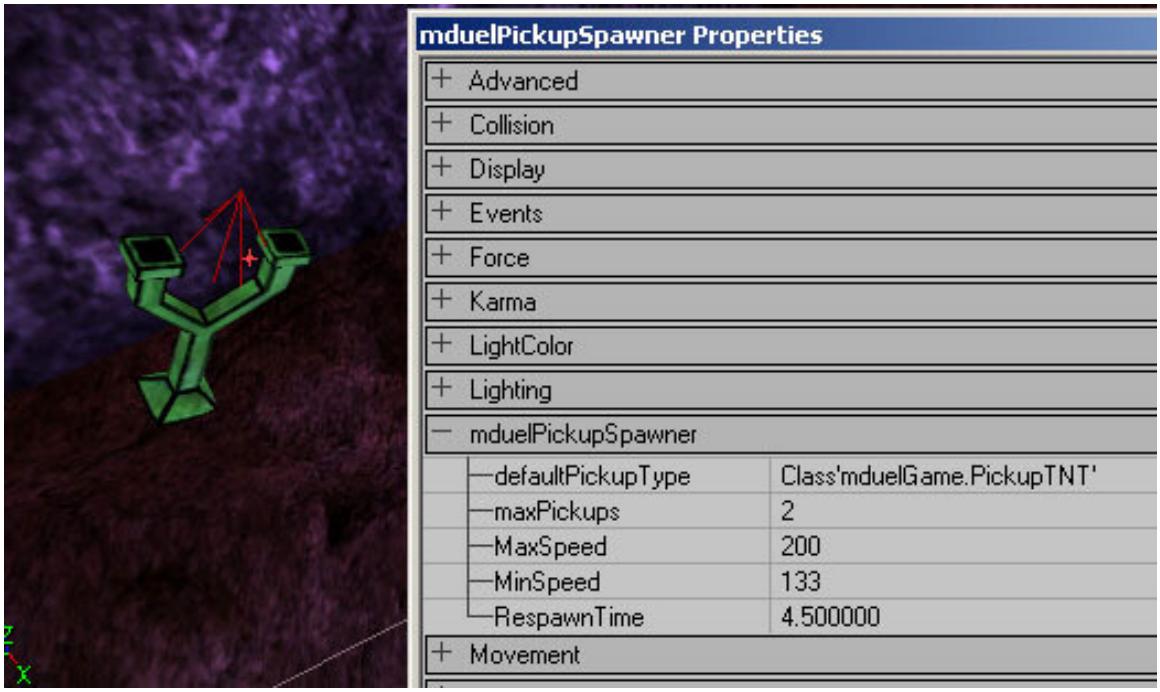
Bad collision

There is an additional type of smashPlatform, known as the smashPOWform. Youll find it under smashPlatform in the actor class browser. This is a bit of a gimmick item, and we really recommend that you don’t use it unless you really know what youre doing and are sure it won’t render your map unplayable. It works the exact same way as a smashPlatform, except that destroying it will destroy every single smashPlatform in the level.

PICKUP SPAWNERS

Pickup spawners are an important part of your mduel map. They are little metal fork things from the original Marshmallow Duel, which (as is obvious) spawn the floating pickups in the game. You place these in the same way as smashPlatforms, the actor name is 'mduelPickupSpawner' and it should be in the list one above smashPlatform.

You can put them wherever you want, but you might want to consider their relation to player starts and fairness etc. Or you might not, if you don't like being fair. Once placed, the rotation of the pickup spawner will have a significant effect on which direction the pickups fly out in, although it is somewhat random. In 2d maps, you will want your pickup spawners on the same plane as your playerstarts at y=0, and you should keep them pointing along that plane and then the pickups should not float out of the player's range.



You have several options for customising pickup spawners. To edit them, right click the spawner you have placed, and go to the mduelPickupSpawner submenu.

defaultPickupType is used to change the pickup type that the spawner will produce. If you set it, it will only spawn that pickup type. This can be a powerful feature for controlling what happens in your map, but it should not be overused, as we have tried to leave the gameplay rather open ended. In the game settings, players can set the types of pickups they want in a game. This does not override the defaultPickupType settings however. This sortof thing is useful if you want to make a custom type of map. For example if you were making a Marshmallow Duel hockey map, with pucks and shields, you would only want a spawner producing pucks, and another producing sheilds, giving you enough control to set up the kindof gameplay you want. In a normal straightforward mduel map you wouldn't really want to use it, unless you are using them in hidden areas where you want a certain type of pickup to be available.

maxPickups sets the maximum amount of pickups. Usually you want this setting on 1, as we allow players to set the pickup multiplyer in the game options, from 1 to 10, to set the amount of pickups they want each spawner to produce. If you need a spawner to produce heaps of pickups in a certain area, you might want to raise the maxPickups number. The best thing to do though is try and design a map which is playable with sparse pickups when they are set to 1, with spawners maxPickups set to 1. The recommended number for the multiplyer is usually 3, and you can suggest this in your level summary. This allows players to set huge amounts of pickups if they want to play the game like that, which might make it easier if they have trouble catching them in 3d maps.

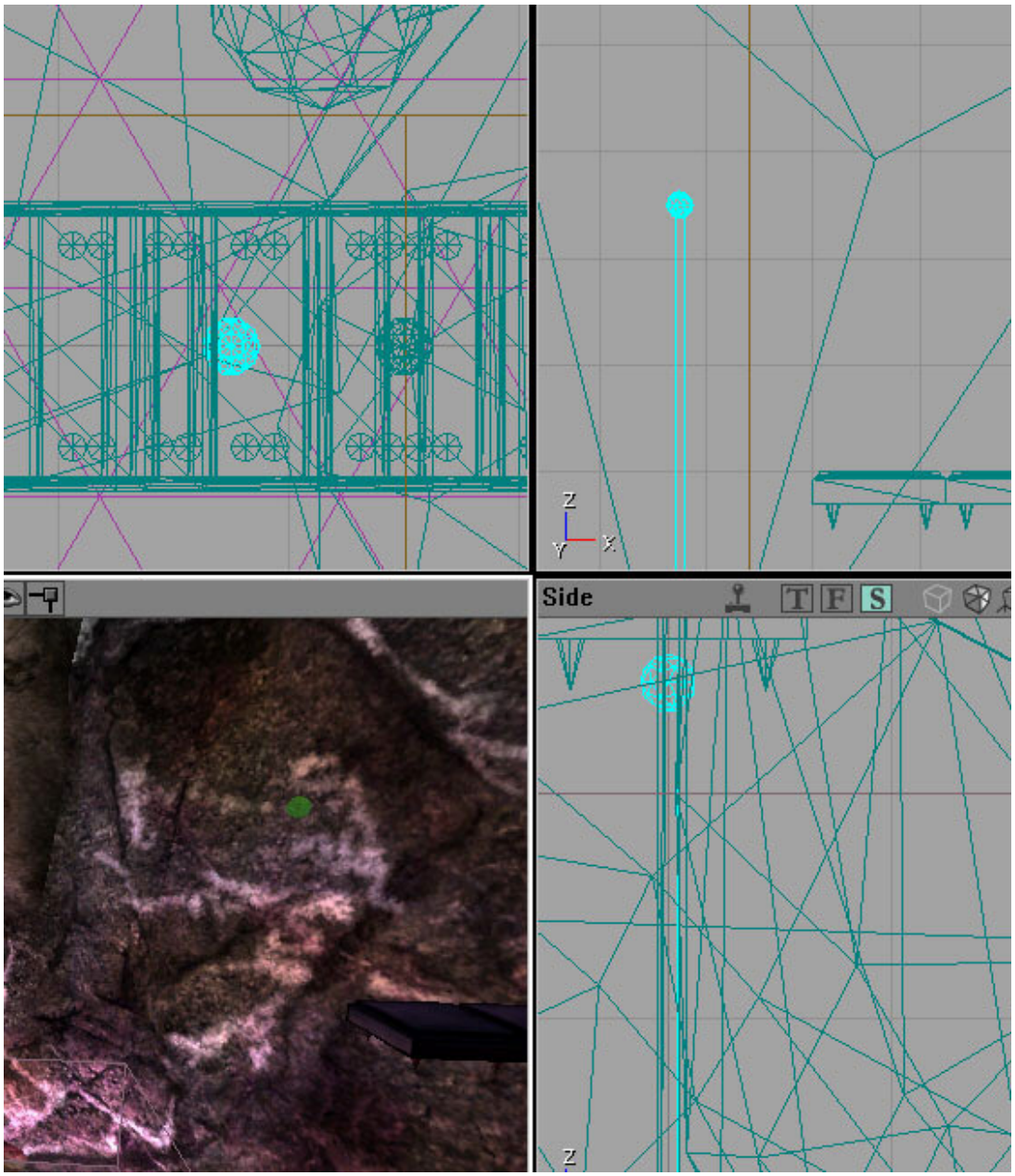
maxSpeed and minSpeed simply control how fast the pickups will move in the game. The spawner picks a random nubmer between the min and max and that sets the speed of the pickups. So the closer the settings, the less variance in pickup speed. The magnet will speed up pickups when it attracts them, however they will eventually slow down to their original speed.

respawnTime sets the time between spawning pickups when your spawner is making more than one. New pickups are always spawned once the old ones die.

CLIMBABLE ROPES

Climbable ropes are a pretty straight forward concept, but can be tricky to get into the right position as they wont show up in the 3d viewport. Like everything else, they are an actor, so select 'climbableRope' from the actor browser and right click and place them in the map. It should be a little above mduelPickupSpawner in the actor list. You will get a short rope, best viewable from the front and side viewports. To make it longer, simply change the drawscale property on the Z axis, and line it up with your platforms or wherever. In the top view you will only see a small square.

To make the ropes look less ugly when they finish in the middle of nowhere, we usually use a climbableRopeCap, which is just a little ball that caps off the top of the rope. You can use the default static mesh from the mduelGameStatic package, or use your own.



A capped climbable rope in 4 different views, the rope itself wont show up in 3d viewport.

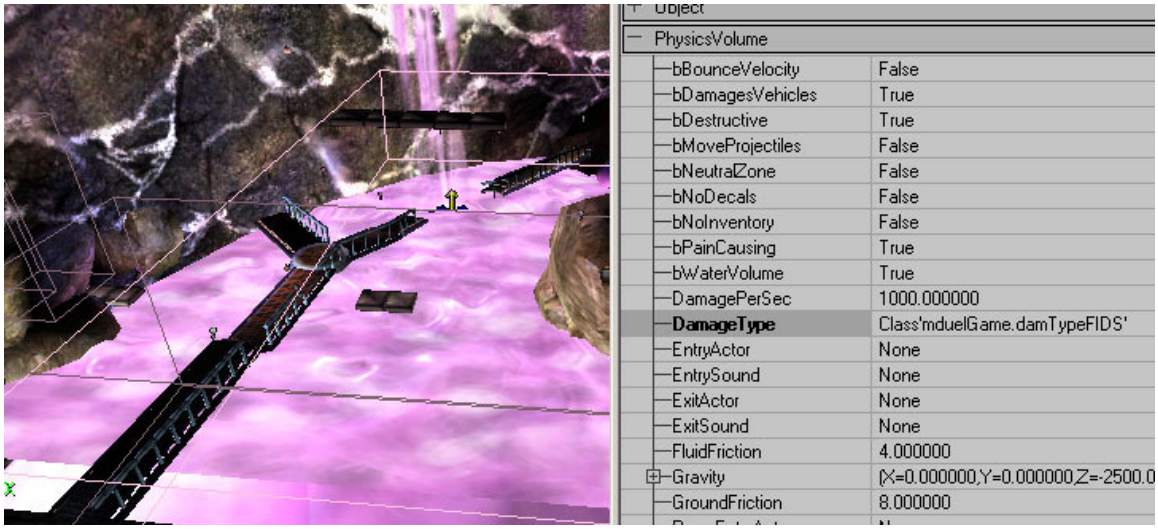
There are several configureable properties for climbableRopes, but there isnt much you'll want to change (unless you want them to break). Firstly, you can't change the static mesh of the rope or you will break it. You can change the skin of the rope, but changing the *skinType* setting. At the moment you have a choice between Rope, Vine, and Cable. Custom types of skins at the moment can not be made by the end user. You wont actually be able to see the effect of changing the skin type untill you view the map in game.

You can change the actual direction of climb on the rope, however it can be tricky to do. You will need to rotate the actual object, and also set the *ClimbDir* setting to the right rotation, or your players will climb along the rope and then fall off where their angles start to differ too much. This could be useful if you want to have several ropes converging at the top of an area, as if they are held back at the corners instead of simply hanging straight down. Or if you are trying to make some kindof crazy confusing map. Changing the climb direction is not recommended, unless you really know what you are doing.

extraDist is something you probably wont want to change. The default is 5, and this represents the area around the rope which the player can grab onto.

MARSHMALLOW

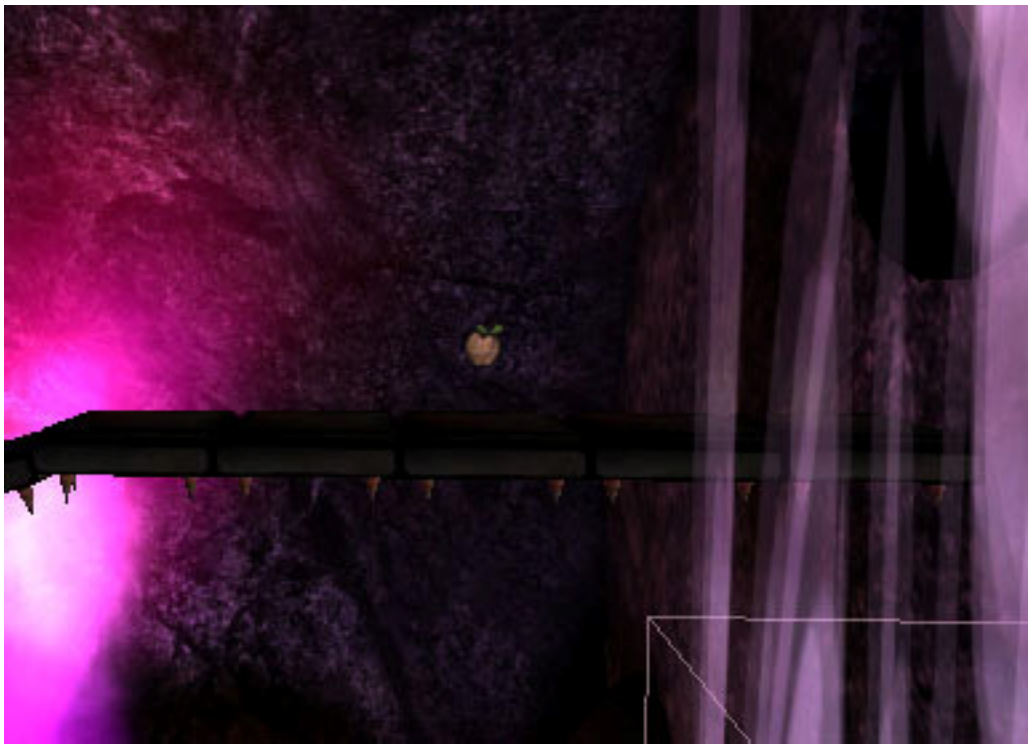
Being Marshmallow Duel, it is typical to have a big pool of Marshmallow at the bottom for players to fight over. This is usually done with a straight forward fluidSurfaceInfo. You can use the Marshmallow shader that comes with the mod, from the mduelTerrainTex package, or you can make your own. The Shader is set to be double sided, so you wont see through the Marshmallow once you go under and FIDs. It is also important to set a physicsVolume with damage, or a killZ in your level and zoneInfo properties. With the physics Volume you can also set gravity and friction settings to make it appear like you are floating in viscous Marshmallow. However when using the physicsVolume to kill the player it is important that you only set the DamageType to damTypeFIDS, fell or drowned, as other settings can give wierd unwanted effects.



A physics volume under a fluid surface with correct damage type set

THE WARP PICKUP

The warp pickup in the original mduel game allowed players to warp to random locations. Obviously UT is much more complicated than a 320 pixel 2d side scroller, so the new warp pickup doesn't pick random locations. Instead, the warp pickup will pick randomly from 'pathNodes' placed in the map. PathNodes are in the actor browser, and should look like a little delicious apple when placed.



The delicious pathNode apple set to warp player onto smash platform

This is an interesting way to do it over randomness becuase it allows you to put them in interesting places. You might want to place the warps in hidden areas, or over smashplatforms which may or may not be there when the player warps, or you might want to be mean and place them right above the Marshmallow. Be creative with your warps and try to make it interesting.

It’s probably not a good idea to run the ‘Build All’ command once you start dropping these in, as it will calculate your bot pathing which is really unnecessary since MD:MR doesn’t support bots at this time.

BLOCKING PICKUPS AND THE PLAYER

Blocking volumes can be used to block the player, to keep them within the defined field of play, but you can also use them to block the pickups and keep them from floating away into the distant heights of the map. You may have areas you want the player to reach, IE to be able to jump high at the top of the map, but place the blocking volume for the pickups slightly lower so that you keep them in more reachable areas.

To set a blocking volume to block pickups, set the 'bClassBlocker' to True, and set a BlockedClasses item to 'mduelPickupBase' (select from the drop down list). mduelPickupBase is actually the base class of all the pickups. If you are trying to do someting creative with blocking volumes you could also block individual pickup types.

Blocking volumes should also block the grappling hook projectile as well, stopping players from climbing to the very top of the map if you don't want them to do that (ie clmbing to the fake backdrop in an open sky map) . If you have any trouble with this working you can also just set the blocked class to 'projGrapple'.

It is also a good idea to put a blocking volume in the Marshmallow Pit so that pickups appear to bounce off the Marshmallow instead of dissappearing into it and reappearing later. This is easy to do, however make sure that 'bClampFluid' is set to False, or else it will mess up your fluidSurfaceInfo.

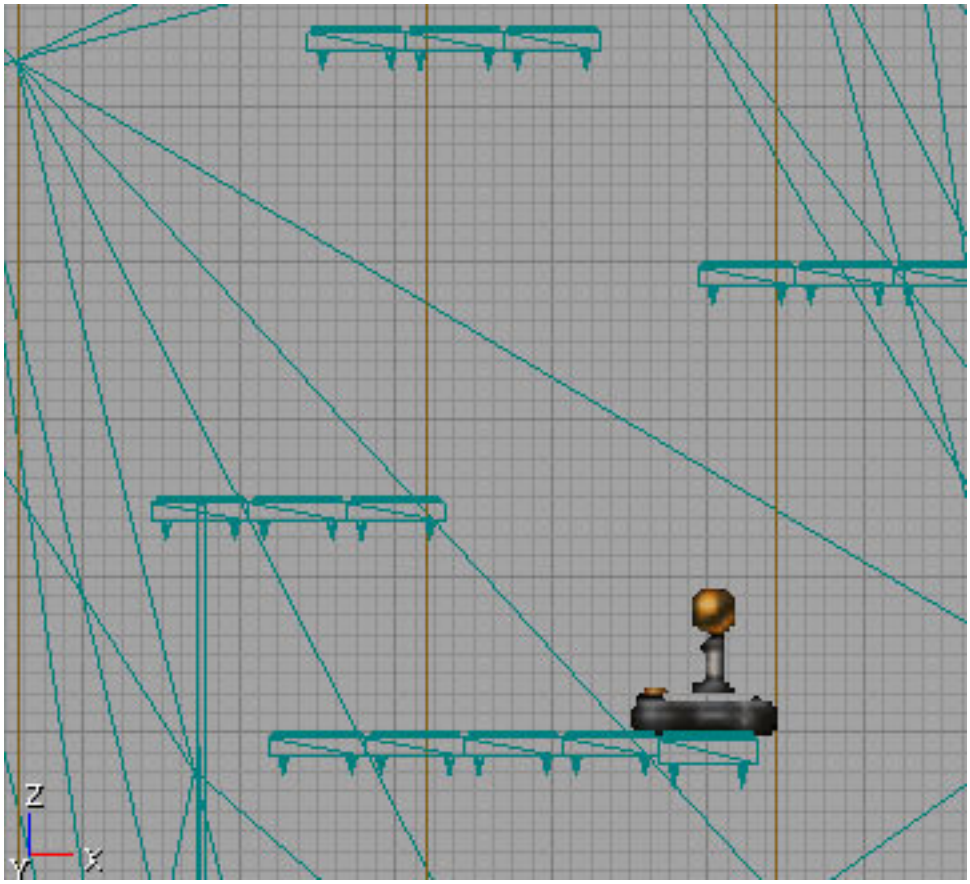
PLATFORM SPACING

When placing your smashPlatforms, you will need to consider the usual jumping distances of the player. Usually a player can jump comfortably from one platform to another about 192 units above it. However if you have 2 layers of platforms, the player will also hit their head on a platform this close above it. It's important to make platforms a decent distance away when jumping upwards, as if they don't reach their full jump height in time, they may hit the platform and roll back, which might make them unstable and fall into the drink. Of course, you might want this to happen i dont know.

Platforms need to have flat top surfaces, becuase it makes it difficult for the game to know if the player can roll or not on the surface if it has a curved or angled edge. You can tilt platforms, although its probably not a good idea to have them to steep or the player's movements might get confused.

If you are making a larger platform out of smaller ones next to eachother, as in most of the Marshmallow Duel main maps, it is best to keep the smaller platforms touching eachother, without gaps between them, as this could interfere with the puck, and other algorithms in the game.

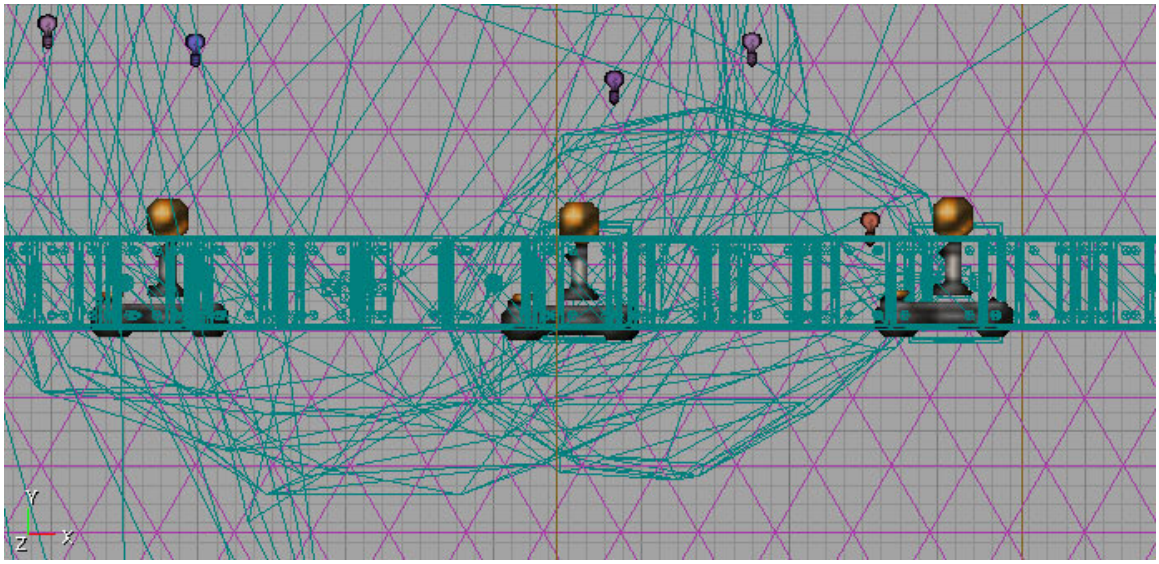
When you have a climbable rope, climing from one platform to platforms above, you have to be careful that platforms at either side of the rope are not going to hit the head of the player when they are climbing up, as it can prevent them from getting up. Just leave a space of at least 32, maybe up to 64 either side of the rope to allow the player up. The rope should also extend some distance above the platform they are trying to reach, so that they can jump down from the rope onto the platform.



Some platforms placed in a 2d map, no gaps between platforms, enough distance between rope and platform, jumping height correct, and player wont hit head on top platform when jumping from bottom platform to middle platform. If you use a grid size of 16, 1 and a half dark grid squares is a good height between platforms.

2D GAMETYPE

As mentioned breifly before, in the 2d gametype everything needs to be placed on the same plane, at least anything that the player is supposed to interact with. Player starts should be placed along the x axis at y=0, at any x or z location, same as pickup spawners, ropes, and the part of a platform you want the player to walk on. You can of course have plenty of things going on in the background to make the game more interesting, and you can have platforms running against this plane for effect.



Top view of 2d map, everything in a line on the right plane. Slight variance of player start y position shouldn't make a difference, so you don't have to be too pedantic.

When making your 2d map, you will need to carve out enough room in the front of the map so that you can see what is going on, if you have any walls in the way in front, you might not be able to see the action. I think the camera is at least roughly 600 units away from the plane where all the action is. You can put objects in the foreground, which can add interesting detail to your map. Sometimes when the player dies, he might move somewhat off his normal plane. You might want to allow for this so that the view does not go wierd when it moves back even further. You can just cut out more space. You also need to take care when players reach the sides of the map, as their view can extend further around the side. If you have anything out the side such as the edges of a fluid surface, they might be able to see it, so you might want to cut away a wall in front of the viewing plane.


2d map with extra space cut away in front of main playing area.

Marshmallow Duel also provides a fixedCamera2D actor to use in your 2d maps. Select it from the actor browser and place somewhere in front of your 2d plane. You can put the camera wherever but it may prove unplayable if you can't see platforms which are too far away from the camera. This is why the fixed camera is suited more for small 2d maps. The rotation of the fixed camera actor represents the view of the actual camera in game.

RANDOM LEVEL GRIDS

There is an additional actor at Actor/RandomLevelGenerator2D which you can use to randomly generate grids of smashPlatforms and climbableRopes. You can make an entire level out of one or more of these, or use them as small sections of randomness in a larger level. This actor is recommended only for 2D levels as it will only generate a plane of platforms.

Simply place the actor into the level and rotate it to the direction you want the grid to face. The actor will serve as the bottom corner of the grid, with the tiles coming out in the direction of the rotational arrow.

The actor comes with many options to configure the grid. They are as follows:

platWidth
The width of the platform staticMesh in UUs. With the default mesh, this is 80. Youll need to change it if you want to use a different model for your platforms.

platVSpacing
The vertical distance in UUs between consecutive levels of the grid. The default is a good value as it allows players to jump up to the next level. If you have very thick platform meshes or just want to change things up a bit, modify this value.

minSpan & maxSpan
Minimum and maximum number of platform meshes to have in a composite platform. Platforms will be made up of a random number of smashPlatforms between these two values.

numRows & numCols
numRows is the number of levels that will be in the grid. The overall height of the grid will be numRows * platVSpacing. Similarly, numCols is the number of tiles wide the entire grid will be. The overall width is numCols * platWidth.

rowFullness
The cutoff at which rows will stop being filled, as a percentage. Think of this as the average density of the grid.

reservedCols
Use this handy array to specify columns that will never get platforms or ropes placed in them. Handy for when you have non-random ropes hanging through the field for a sure place to climb up or down. The numbers specify the *n*th column from the generator actor, where 0 is the first one.

platformTime
How many seconds the platforms should stay hidden for if destroyed.

platformMesh
The staticMesh that all the platforms will use. Remember to set platWidth if you use a different one than the normal mesh.

minropes & maxRopes
The min and max number of ropes that will be in the field. Ropes will never exceed their capacity or get placed too near to other ropes as specified with ropeSpacing, so note that you will not get as many as you specify if you expect too high a number.

ropeSpacing
Number of blank tiles between each rope.

minRopeHeight & maxRopeHeight
Min and max values for the height of the ropes, as the number of vertical spacings to span.

ropeSkinType
The skin style of the ropes, as you would normally specify when placing ropes individually.

The End.

Chris Grist
24 Oct. 2005