Red Orchestra
Software Developer's Kit [SDK]
Manual

Table of Contents

Part 1: Introduction to the Level Editor

1.1 Introduction

If you are reading this document then it is likely you are considering entering the world of level development for Red Orchestra: Ostfront '41-45. Level design can be a highly rewarding experience, but it can also be hard work. Perhaps the most difficult aspect of level design is climbing the learning curve to get to the point where you can begin to see your vision realized in game. That's where this document may help.

I have created this tutorial with the intention of helping smooth over some of the initial bumps encountered when starting work in level design. This tutorial will not make you an expert level designer, but it will walk you through the steps of creating a simple, basic, but working Red Orchestra: Ostfront '41-45 level that you can actually play. If you follow it through to the end you will be left with a useful test level that you can use to experiment further with Red Orchestra mapping techniques.

The tutorial is really focused on those that have little or no experience with level editors and those with some knowledge but who are new to the Red Orchestra world. Veteran level designers, including those who have produced Red Orchestra maps in the past, may still benefit from the overview of basic map actors and find the included, completed test level useful for experimentation.

So, without further ado, let's move on to the first part of the tutorial.

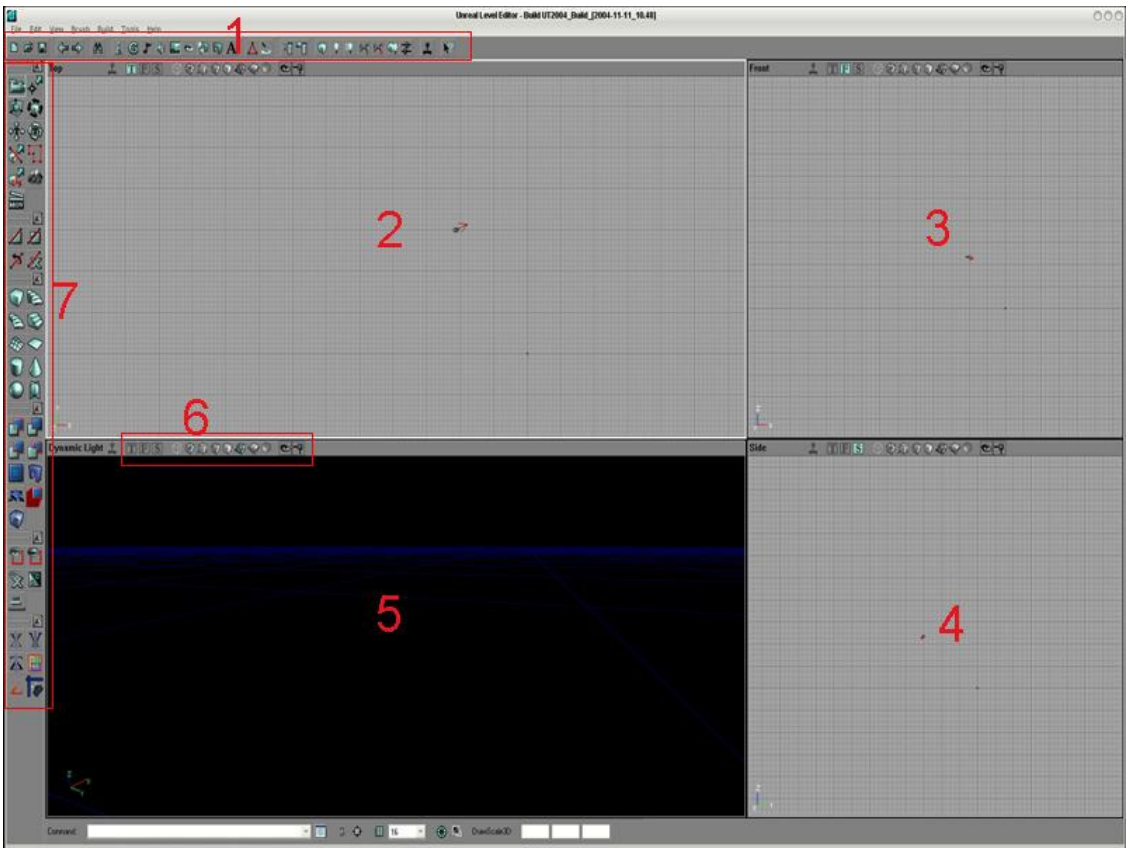
1.2 A Few Words About The Red Orchestra Level Editor

Before we begin, I'll take a little time to talk about our primary tool for level editing: the Red Orchestra level editor. I'm not going to try to teach you how to use the level editor like a pro, that's way beyond the scope of this tutorial. Instead, I'll give you a brief overview of the basic principles: just enough to follow along with the tutorials.

1.2.1 A Quick Walkthrough

First, let's start by firing up the level editor. You'll find the shortcut for the editor in the system folder under your Red Orchestra: Ostfront '41-45 installation.

Once the editor has started you should see something like this:



As you can see, the editor is broken down into four rough quadrants with some supporting toolbars and palettes. The parts shown are:

1. The main toolbar.
2. The top view pane
3. The front view pane
4. The side view pane
5. The rendered view pane
6. The view pane toolbar, present in all panes.
7. The tool palette

This is the default arrangement of the editor. You can change this arrangement to suit your style, but we won't go into that in this tutorial.

You'll do most of your work in the top, front, and side view panes. The rendered view pane is where you will be able to see the results of your work.

You can navigate about your level in the rendered view pane by using the mouse. You can move forward and backward in the view by moving the mouse around while holding down button 1. You can rotate your view in the rendered pane by moving the mouse while holding down button 3. Finally, you can move up and down in the rendered view by moving the mouse while holding down both buttons 1 and 3.

The tool palette on the left side of the editor is where you will find most of your tools.

All right, that was a very quick overview of the editor workspace. I'll explain a bit more about the various features of the editor as the tutorial progresses, but let's move on at this point. I think most people learn best by actually doing so we'll try to get to a point where we can do something as quickly as possible.

1.2.2 Carving Out Your Own Personal Space

The Red Orchestra level editor employs a subtraction paradigm. That is, when you open the editor the world is like a solid mass of clay. Levels are created by "subtracting" spaces from the clay mass and then adding stuff back to fill it out with details. One of the first things we will do when we start the tutorial is to carve out an area in which we can build the rest of the level.

1.2.3 Building Objects

In general, you will construct a level using three types of objects:

1. BSP (or Binary Space Partitioning) objects. BSP objects can be walls, roofs, stairs, even terrain. The editor provides a number of tools for creating and manipulating BSP objects.
2. Static mesh objects. Static meshes are objects created outside of the editor, usually in a 3d modeling application such as 3ds Max or Maya.
3. Map actors. These are objects that essentially hook the level up to the game engine. They control a variety of things such as lighting, sounds, events, and game objectives.

BSP objects are manipulated in the editor using what is known as brushes. Brushes are three-dimensional shapes that represent templates for adding or subtracting from that large clay mass of the world.

1.2.4 Brushes

As I just explained, brushes are used to add and subtract BSP objects from the world. The Red Orchestra level editor provides a number of preset brush shapes for level designers to use. These presets are usually all you'll need to get the job done. If not, the shapes can be edited to suit your need (but that's an advanced topic).

So let's move on to our first task when starting a new level: creating the world space.

1.3 Creating the World

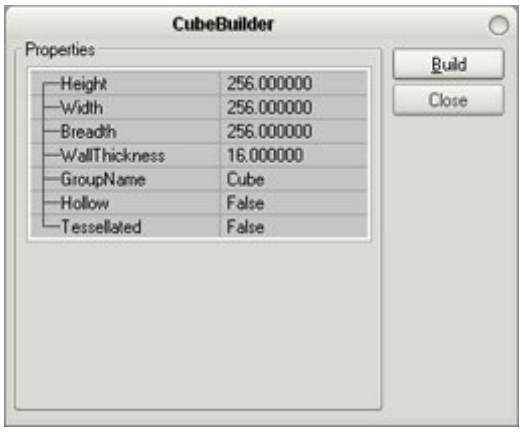
We're now ready to start creating our quick start level. The first thing we need to do is to create a space where we can build the level. As we mentioned in the preceding section, The Red Orchestra level editor works using a subtractive paradigm. Right now, our world is entirely clay. We need to carve out a space to work in. We'll do that using a very large cubic brush.

First let's start with a fresh workspace. Go to the File menu and select "New". This will create a new level for us.

Once that is done we need to create a brush to use to carve out the world. We'll use a cubic brush. Our brush palette is found on the left hand side of the editor and looks like this:

As you can see, there are a number of brush shapes that are available to you, including cubes, cones, cylinders, as well as stairs, terrain and other odd shapes. We only need a cube so we'll use that brush (found in the top left corner).

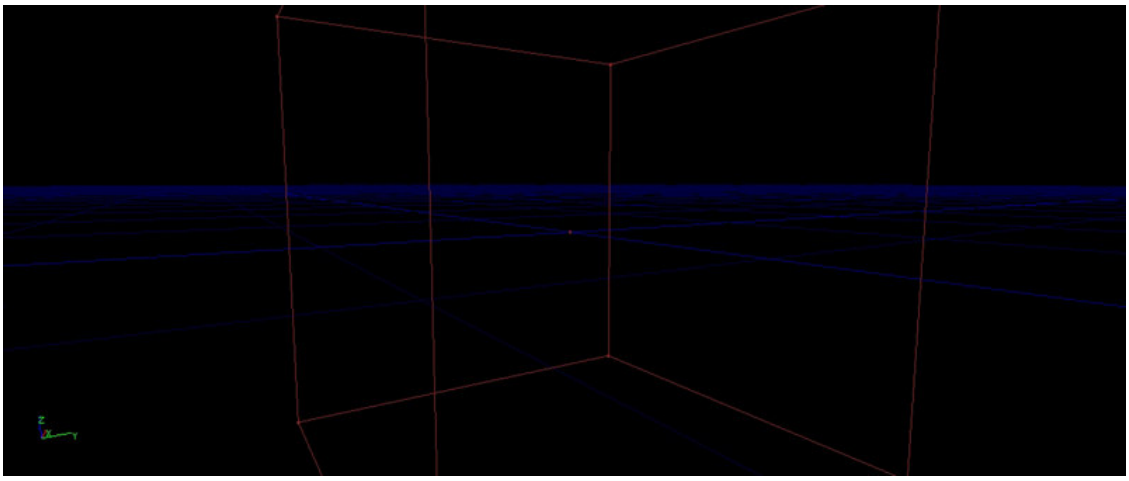
We'll need to configure the brush so it's the correct size. The Red Orchestra level editor allows us to do that by right clicking on the brush. Right click on the cube brush now. You should now see the following dialog:



This dialog allows you to control the properties of the brush. The default values for a cube brush are currently displayed in the dialog. Right now, the only properties that concern us are the ones for the height, width, and breadth of the brush. Currently, these are set to 256 units each. This is barely larger than a player (a player in Red Orchestra: Ostfront '41-45 is roughly 128 units in height), which will clearly not do. We need something much, much larger.

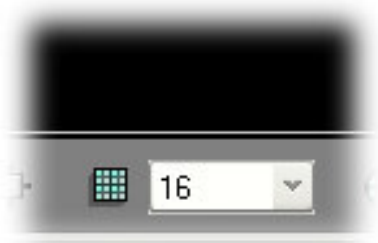
Select the height, width, and breadth properties in turn and set them to 4096 each. Why 4096? Well, the editor units are based on base 2 numbers (256 is 2^8, which is 2 multiplied by itself 8 times). 4096 is 2^{12} and it's large enough for our purposes, so we'll use that. Don't worry; you'll get use to thinking in editor friendly numbers very quickly.

Once we've set the brush properties to the values we want, we can go ahead and build the brush. Do this by clicking on the "Build" button on the CubeBuilder dialog. You should see a large red cube appear in the rendered view pane in the editor, similar to the picture shown below (you may have to zoom out to see it):



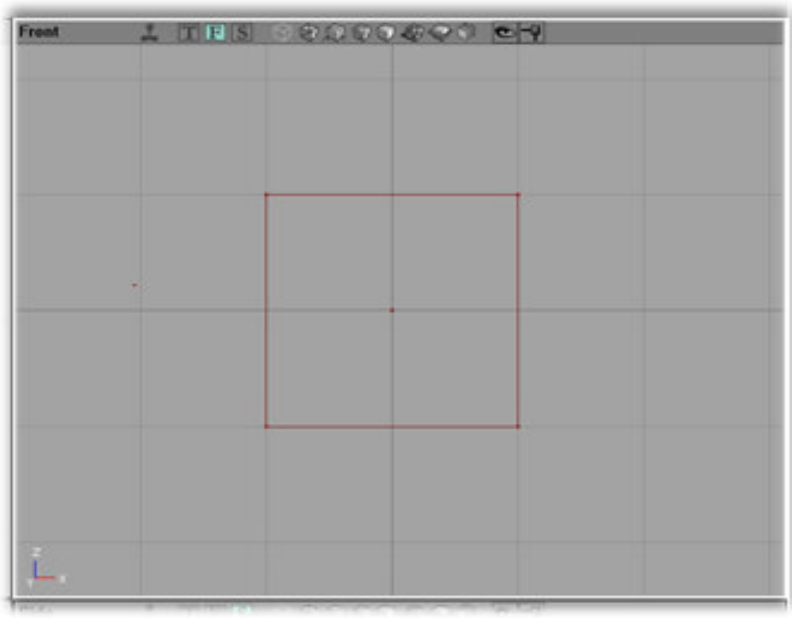
We're not quite ready to carve out our world yet, however. Right now, the brush is centered on the middle of the world. It's more convenient for us (not to mention a learning experience) to move the brush a bit. You don't technically have to do this, but I find it easier.

Zoom out in the front view pane until you can see all of the brush. You'll notice the grid lines in the view. The resolution of these grid lines can be adjusted by the grid control selector found at the bottom of the editor.

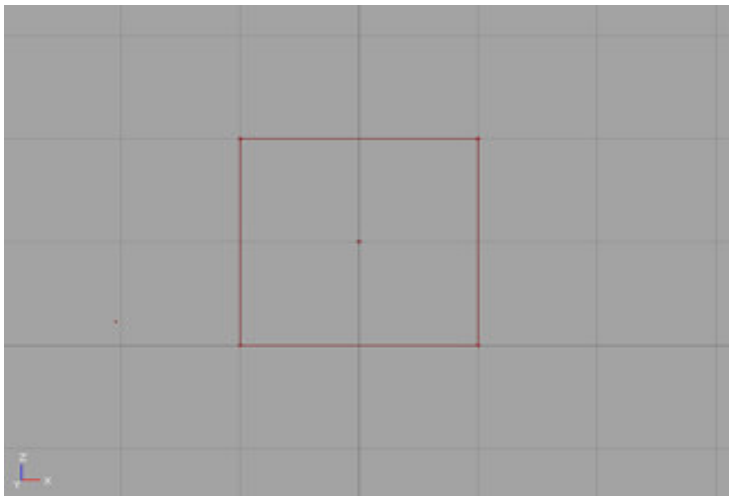


Let's change the scale on the grid to 2048 units so we can see more clearly. Click on the drop down control and select 2048.

You should see a change in all of the editor view panes. The front view pane should now look something like this:



Lets move the brush up so that the bottom sits at the middle of the editor coordinate system. We can do this by selecting the brush and moving the mouse up while simultaneously holding down mouse button 1 and the shift key. After this is done, the front view should look like this:



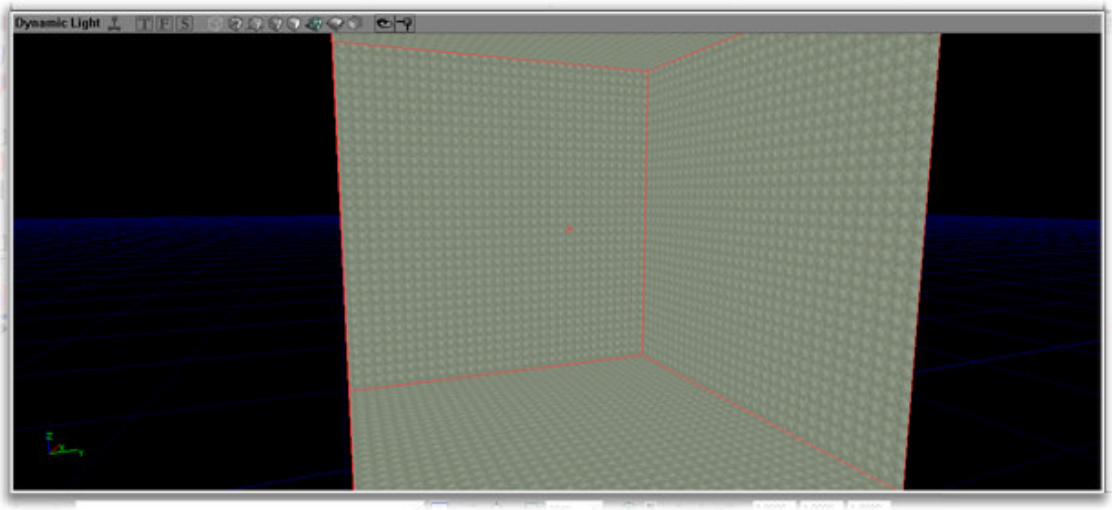
A subtle difference, but one that will help later. You can now reset the grid scale to something sensible, like 16.

We're now ready to carve out our working area. The command to do this is found on the tools palette, shown below:

The circled button is the subtract command. This action subtracts the volume inside the brush from the original "clay" space and leaves us with an open area in which we can build the rest of our level.

Click on the subtract button now.

Having done this, you may notice a few things. First, the rendered view pane has changed. You should now see a cubic space where previously there was simply nothing. You'll also notice the walls of this cube are covered in a greenish pattern. Your view should look something like that below:



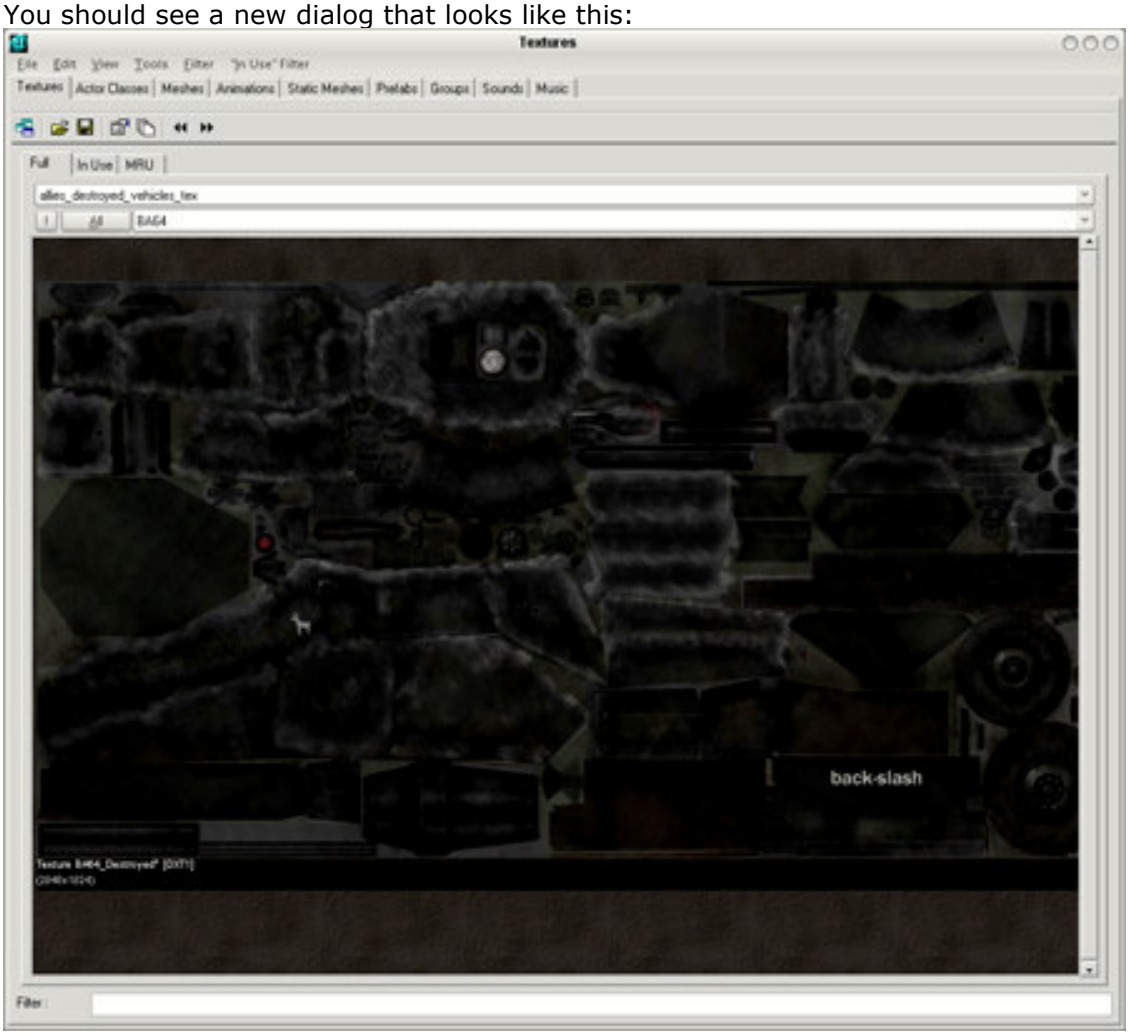
The green pattern is the default texture that the editor applies whenever a new brush is added or subtracted. It's quite ugly, so let's replace it.

First, let's undo that subtract we just finished.

You can undo certain actions in the Red Orchestra level editor by pushing the button in the main toolbar that looks like a left pointing arrow. Press the undo button now. This should return the editor to the state it was in before we pressed the subtract button.

Now let's select a more appropriate texture and subtract the brush again. In the editor, you select textures in the texture browser. This button is also found in the main toolbar and looks a bit like a painting. Click it now.

You should see a new dialog that looks like this:

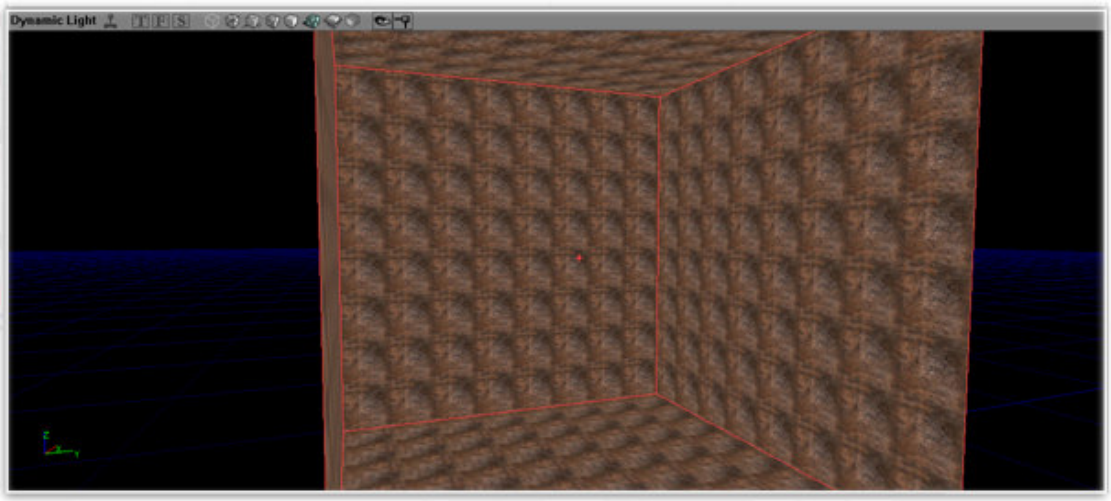


Now lets find a nice texture to use. Click on the open button on the browser toolbar (the open folder icon). You should now be presented with a browser dialog that shows the Red Orchestra: Ostfront ’41-45 texture packages. Select and open the Architecture_T package (Architecture_T.utx file).

The texture browser should now display a series of nice brick and concrete textures. Take the time to scroll through them and take a look. Any texture will do for our purposes, but let’s select the m_brickwork1 texture. Simply click on this texture and then close the texture browser.

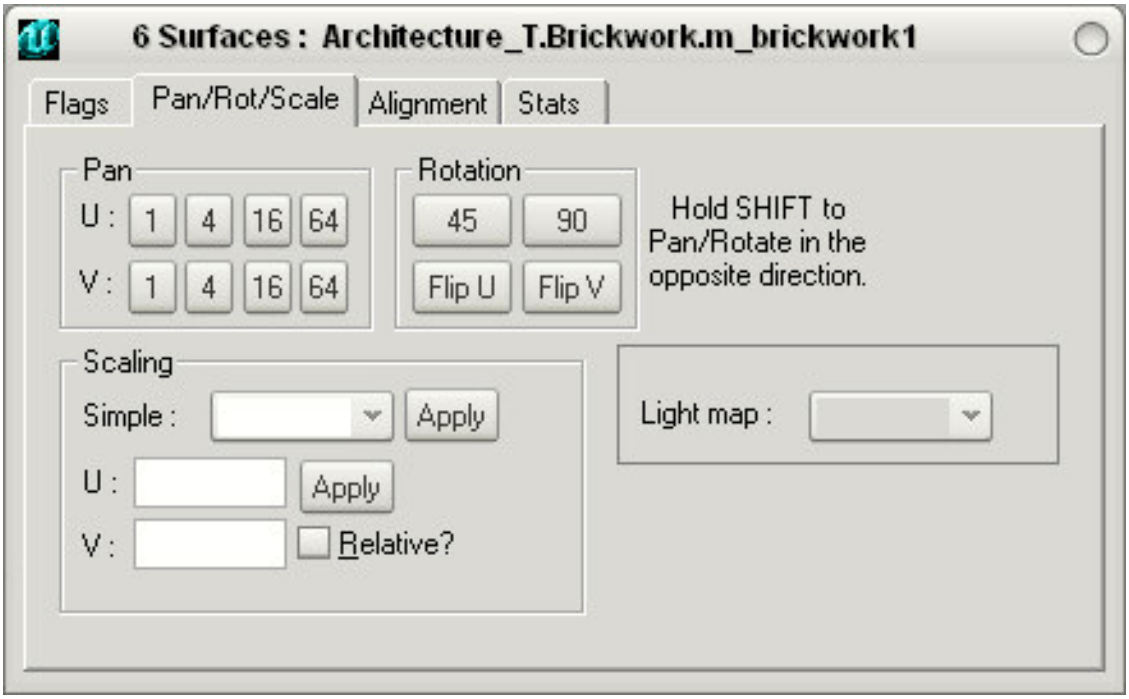
Now we can go and re-subtract that brush again. Go ahead and do that. You should see the same cube space appear in the rendered view pane, but now it should be painted with the new texture we just selected.

In fact, your view should look something like this:

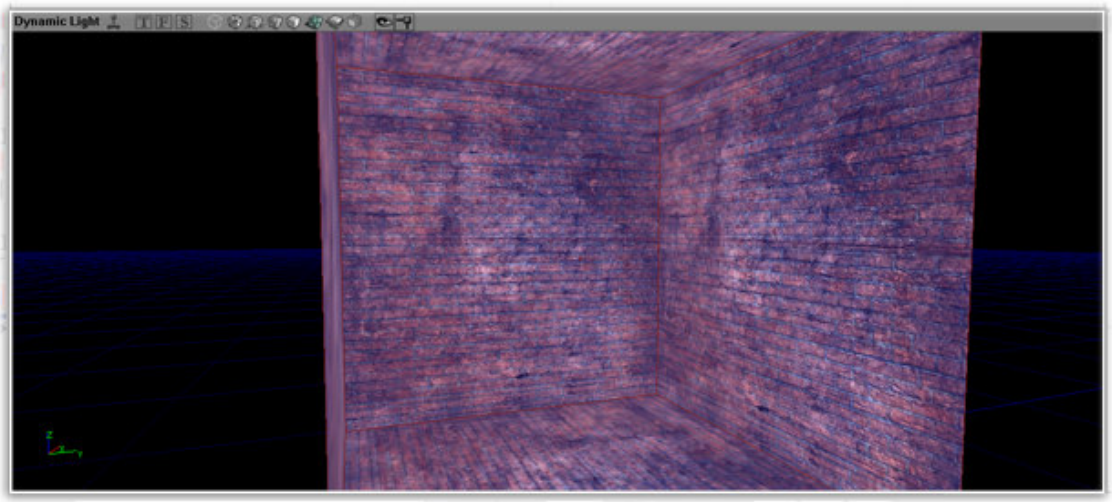


A clear and regular pattern shows on the walls, which is kind of annoying, so let's fix that. We can scale the texture up so that it's larger and repeats less often. To do this, click on any of the walls of the cube. When you do this, you'll see the wall become shaded. This is the editor's way of showing you which surface is currently selected.

Now right click to bring up a context sensitive dialog. One of the options on the dialog is Select All Surfaces. Choose that option and you will see all of the cubes surfaces become selected. Right click to bring up the dialog again and select Surface Properties. You will now be presented with the surface properties dialog. Select the Pan/Rot/Scale tab. You should see a dialog that looks like this:



We're interested in scaling the texture, so focus on the Scaling group of controls. We'll just do a simple scaling on this texture so click on the Simple drop down list. Select a scaling of 8 from the list and hit the Apply button. You'll immediately notice the difference in the rendered view pane.



Ok, so it's not very realistic, but it looks better than before. Let's move on.

Actually, this would be a very good point to save your level. Since we're investing quite a bit of time here it's prudent to save frequently. Save you level and call it "RO-QuickStart" (all Red Orchestra: Ostfront '41-45 levels must start with the RO- prefix).

We can also build our level at this point. All levels constructed in the editor must be built before they can be played. Building a level does quite a number of things but the most important of these is figuring out how to light the level.

The build controls are found in the main application toolbar. Click on the button that looks like a cube and a light bulb together. This will rebuild everything in the level.

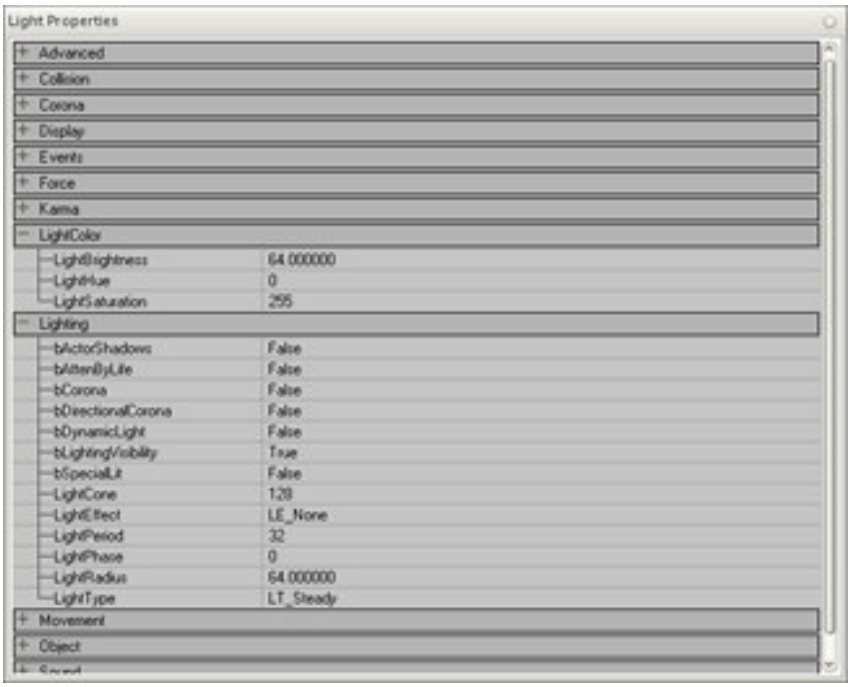
The first thing you should notice at this point is that our view of the cube in the rendered view pane just became quite black. There's a simple explanation for this: our level has no lights. Let's fix this.

Select a point anywhere in the cube and right click to bring up the context sensitive dialog again. Select the option Add Light Here to place a light into the level. You'll see a small light appear in the rendered view pane and a light bulb shaped map actor will be seen in the other 3 views.

Move the light until it is a reasonable distance from the floor of our world cube (let's say 1024 units). You can move the light by selecting it and dragging it to the proper position in any of the views while also holding down the control key.

Once the light is in the correct position, you can rebuild the level again. The light is probably too dim to light up much of the level at this point. We can correct that.

Select the light and right click on it. Choose the light properties entry in the menu. You should now see the light properties dialog. All map actors in the editor have sets of properties that control how they behave. You, as a level designer, can control how a map actor works by changing its properties. You will find yourself doing this quite a bit when working on maps in the Red Orchestra level editor.



In this case we are simply interested in extending the range of the light so it illuminates more of our level. This is controlled by the LightRadius property in the Lighting group. The default value for this property is 64 units. Let's increase that to 256 and then rebuild the level again. Hopefully you see a difference.

The level might not seem like much at this point, but we have just successfully set the stage for our future level building work. Nice job. Let's save the level again and move on to the next section.

1.4 The ROLevelInfo Actor

If I were allowed to generalize very broadly, I could say that work on a Red Orchestra: Ostfront '41-45 level could be broken down into 3 basic types of jobs:

1. Terrain work
2. Adding decorations using either BSP or static mesh.
3. Adding map actors.

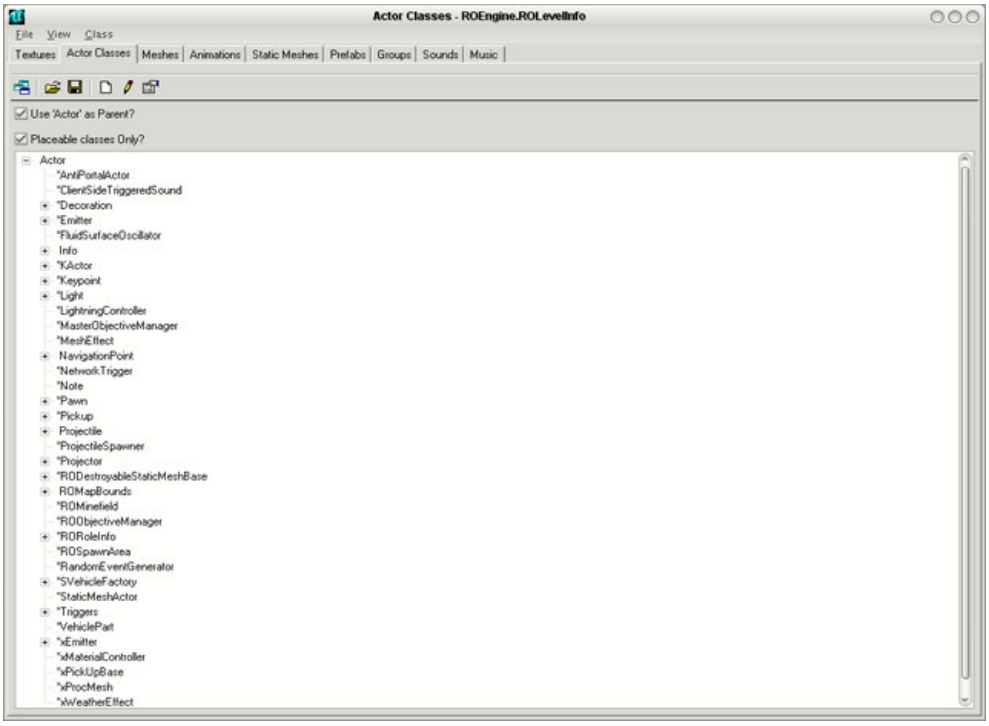
In part 2 of our quick start tutorial I'm going to introduce you to map actors. Put simply, map actors are hooks into the game engine that a level designer can place in their map. These hooks affect the way the engine works with the level objects and includes everything from lighting and game types to player spawning and game objectives.

You will make use of a large number of map actors in any level you make. Most of these actors are associated with the game engine itself. I won't cover these in any great detail in this tutorial. Instead, I will concentrate on those actors provided specifically for Red Orchestra: Ostfront '41-45.

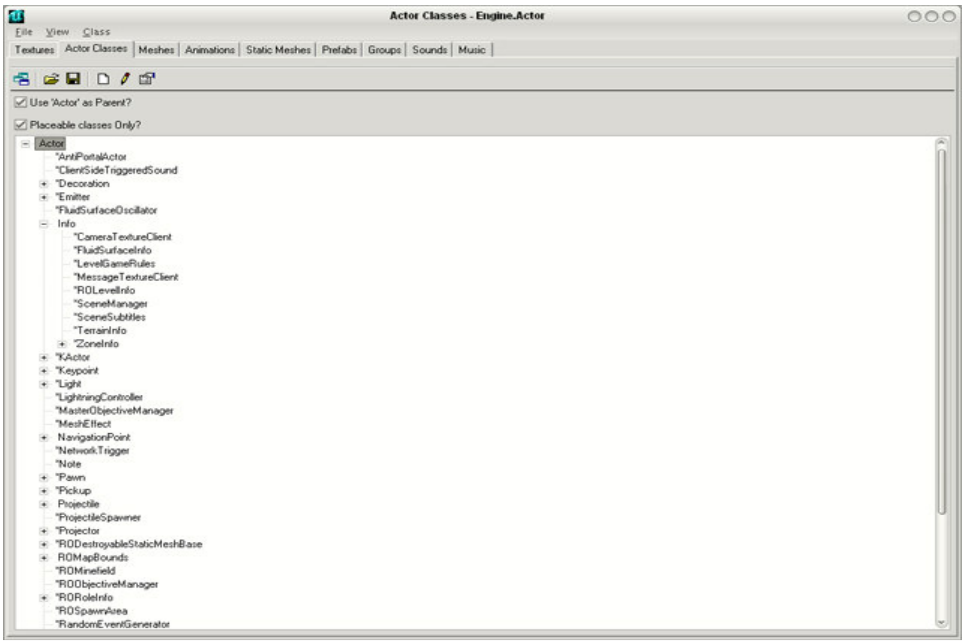
The first, and perhaps most important of these is the ROLevelInfo actor. This actor lets level designers set some basic properties for their map. All Red Orchestra maps require one instance of this actor in order to function properly.

So let's get started. Launch the Red Orchestra level editor and open the quick start map we began in the part 1, if you haven't already. The first thing we'll need to do is find the ROLevelInfo actor in the actor browser. Remember the texture browser in part 1? Well, we're going to use it again. If you don't remember, it's the button on the main toolbar that looks a bit like a painting. Open the browser and click on the tab that says "Actor Classes".

You should see something that looks like this:

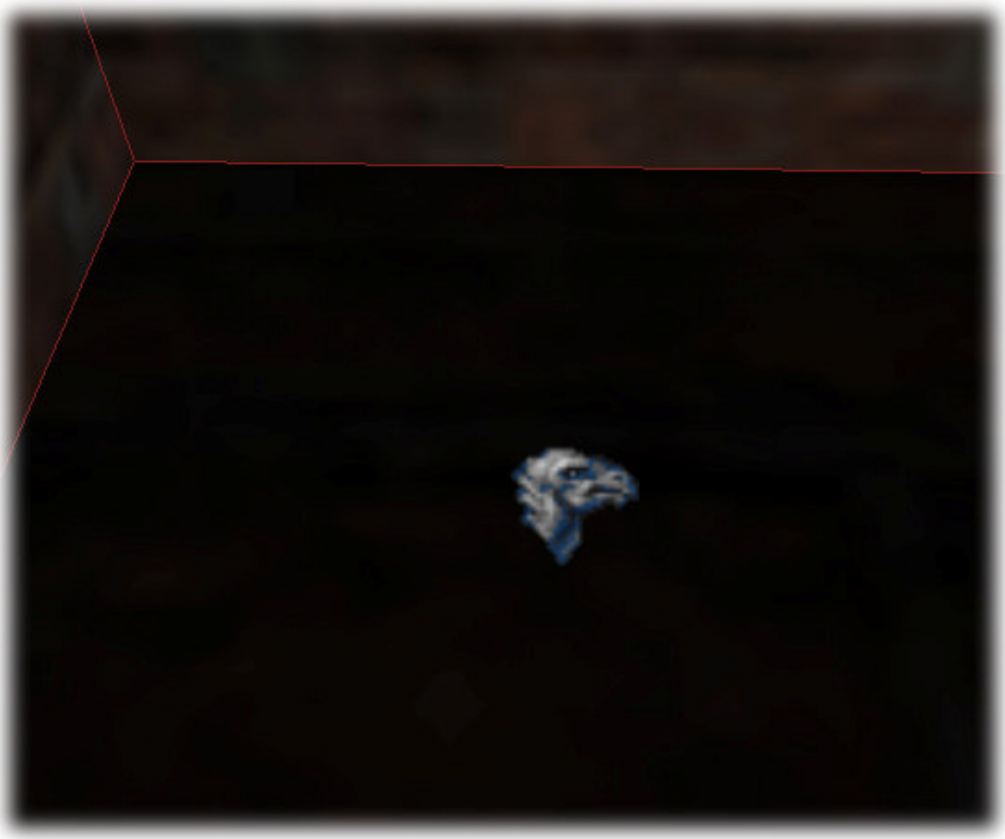


The actors in the editor are arranged in a kind of hierarchy depending on their job. If you look a little way down the list, you'll see an actor named "Info". Beside the actor you'll see a little plus sign ('+'). That sign indicates that there are more actors related to this one. Click on the plus sign. The browser should then look something like this:



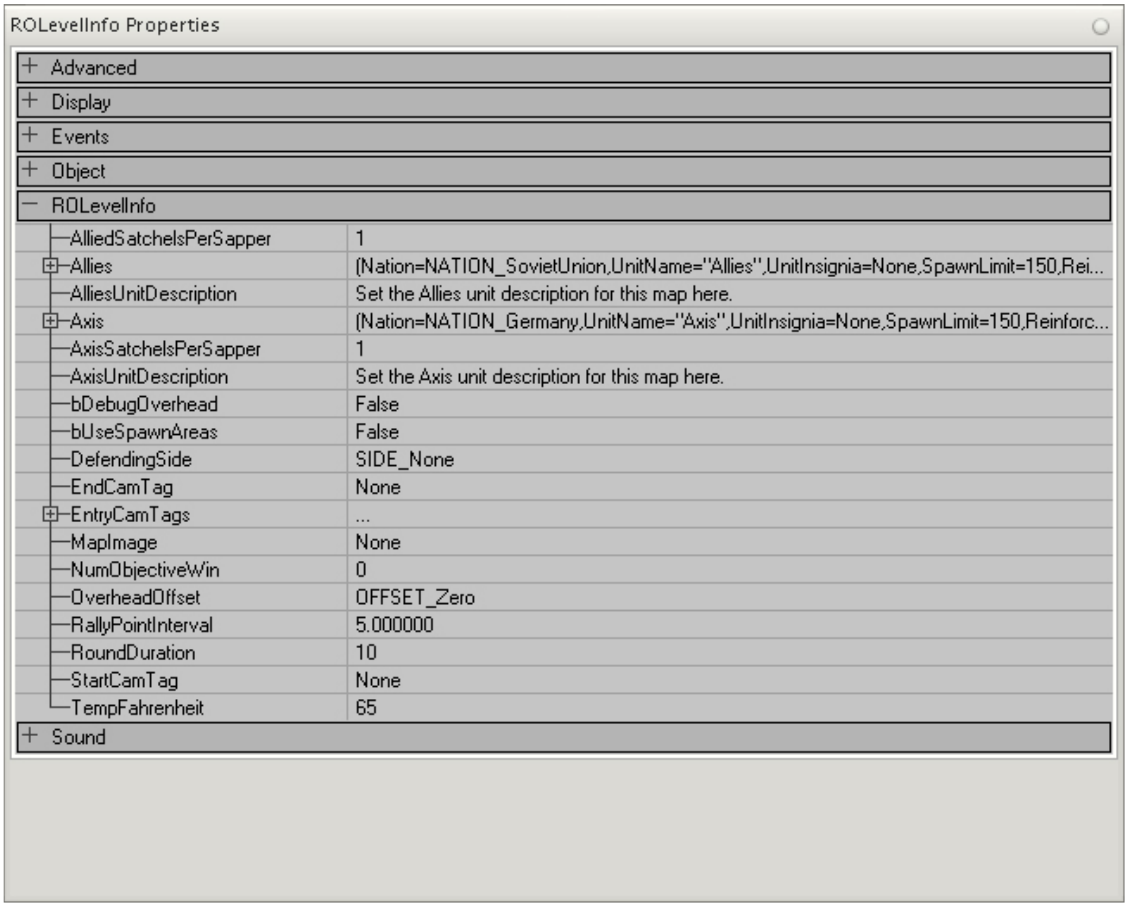
Now, if you look in the new list, you should find what we're looking for. Click on the ROLevelInfo actor and then close the dialog.
Now we can place the actor in the level. The ROLevelInfo actor doesn't need to be situated anywhere in particular in a map, so I usually stick it in a corner out of the way. Find a corner of the map in the rendered view pane and right click to bring up the context sensitive dialog. One of the choices near the top of the list should say "Add ROLevelInfo here".

You'll notice a funny looking icon that sort of looks like a horse's head (I've never been sure exactly what it is. If you figure it out, let me know) appears in your level.



Now we can go ahead and configure the actor. All actors have sets of properties that can be adjusted by level designers to customize their behavior. Double clicking on any actor in the map will bring up its property dialog. Open the properties dialog and select the ROLevelInfo properties group.

You should see a dialog that looks like this:



I won't go into all the properties, but we'll look at some of the more important ones. For this map, I've arbitrarily decided to have the Germans attacking. We can communicate this to the game via the DefendingSide property. Click on the drop down list there and set the property to SIDE_Allies.

We'll only have one objective in the map, so let's set the NumObjectiveWin property to 1. This tells the game that when the attacker captures 1 objective, they win.

Finally, we want to set the bUseSpawnAreas to True. We'll be setting up the spawn areas in the next part of this tutorial.

There're obviously a lot of other properties we could fool with, but it's not necessary if we just want to get a simple level working. For a more detailed look at the Red Orchestra: Ostfront '41-45 map actors and their properties, I'd recommend looking at our reference guide.

You can now rebuild and save the level. It probably doesn't feel like we've done much, but you now know how to add just about any map actor to a level. In the following parts of this tutorial we'll be adding more actors so this is a good, and necessary, first step.

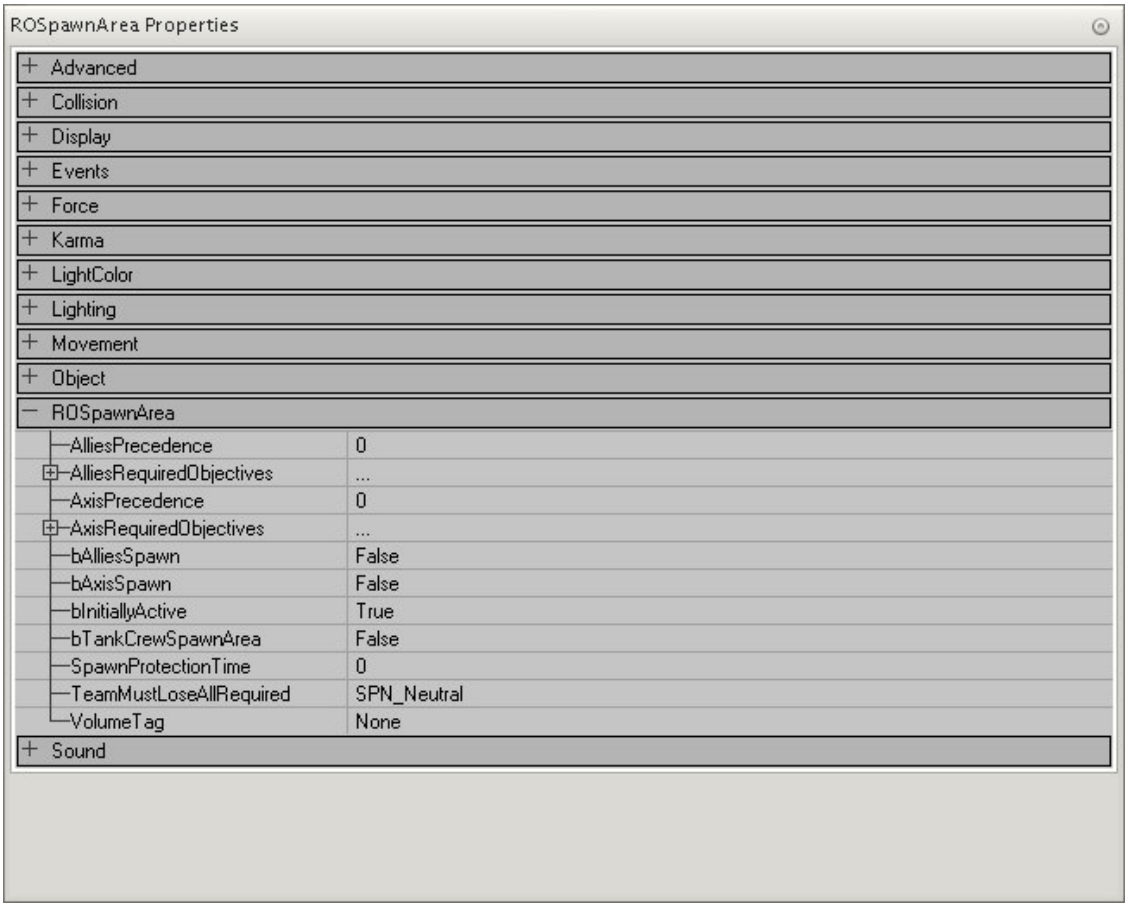
1.5 Introduction to Spawning

At some point we'll actually want to go play the level. We can't do that unless we configure the spawning, so let's do that next.

Spawning is the games way of placing players into a level. Players can enter a level only at fixed locations, known as spawn areas. Spawn areas consist of at least two different actors: a ROSpawnArea actor and a number of PlayerStart actors. We'll need two spawn areas: one for the Germans and one for the Soviets.

First, open The Red Orchestra level editor and load your quick start level. Then bring up the actor browser again and find the ROSpawnArea actor. Let's place the spawn areas in the map, just like we did with the ROLevelInfo actor in the last section. I suggest placing the spawn areas near opposite walls in the map.

Once the actors are placed, we need to configure them. Double click on one of the actors to bring up the property dialog then open up the ROSpawnArea properties. You should see a dialog that looks like this:



Again, I won't go into every property listed here, just the ones we'll need to get started.

We'll make this first area the German spawn so set the bAxisSpawn property to True. Now click on the Events property group. You will see a property called "Tag". Set that to something sensible like "GermanSpawn".

That sets the spawn area, but now we need to add some actual player start locations. Find the ROSpawnArea in the rendered view pane and right click somewhere near the actor to bring up the context sensitive menu. About half way down the list you should see an entry that will allow you to add a PlayerStart. Once you add it you'll see something that looks like this:

Notice that the PlayerStart actor looks like a joystick. The red arrow indicates the direction the player will be facing when they enter the game. You can change this by selecting the actor and moving the mouse while holding down the right mouse button and the control key. Make sure the player start faces the opposite wall.

Now lets configure the PlayerStart. Bring up it's property dialog (by double clicking on the actor) and find the Events properties. We need to set the Tag property to the same value we set in the ROSpawnArea ("GermanSpawn").

What did we just do there? We told the PlayerStart that it is associated with that ROSpawnArea actor. Only one spawn area is active for one team at any time and only the PlayerStart actors associated with the active spawn area will ever be used.

Some confusing things can happen if you don't properly link PlayerStarts and spawn areas, so take your time and make sure it's correct.

Since RO:Ostfront '41-45 supports up to 16 players a side, we're going to need more PlayerStart actors. I would actually recommend adding more than the minimum 16, just in case. You don't need to add them all 16 for this map, but a few more would be wise.

Of course, you can always copy the original PlayerStart. There's easy way to do this. If you bring up the context sensitive dialog (by right clicking on the PlayerStart actor), you'll see the Edit sub-menu. Under that menu you'll see Copy and Paste options. You can copy and paste a few more PlayerStart actors this way. Even easier, there's a Duplicate option further down the menu as well.

Create a few more copies of the PlayerStart actor this way. Make sure they're not too close to each other. You can move an actor around in the editor by dragging it while holding down the shift key.

Once you're done, your level should look a little like this:

That's it for the German spawn area. Of course, that's only half the battle. We need to define a similar area for the Soviets. The process is exactly the same, except this time we'll use the tag "SovietSpawn" in all the actors. Don't forget to set the bAlliesSpawn property in the ROSpawnArea actor to True.

At this point, I would rebuild and save the level. The spawn areas are complete. We're very close to being able to actually play this level. We'll put the final piece in place in the next section of this tutorial.

1.6 Roles

There is just one more small bit of work to do before we can play our quick start level and that is to add a few roles. Roles (sometimes referred to as "kits" in other games) represent different types of soldiers. If you've played Red Orchestra, you're familiar with role selection as part of the process of entering an online battle.

Red Orchestra: Ostfront '41-45 allows you to add the following roles to your levels:

1. Commander
2. Rifleman
3. Assault or Stormtrooper
4. Machine gun crew
5. Combat engineer
6. Anti-tank troops
7. Snipers
8. Tank commander
9. Tank crewman

Each role has specific weapons and equipment. Within limits, level designers can configure the weapons loadouts for a particular role to suit their needs.

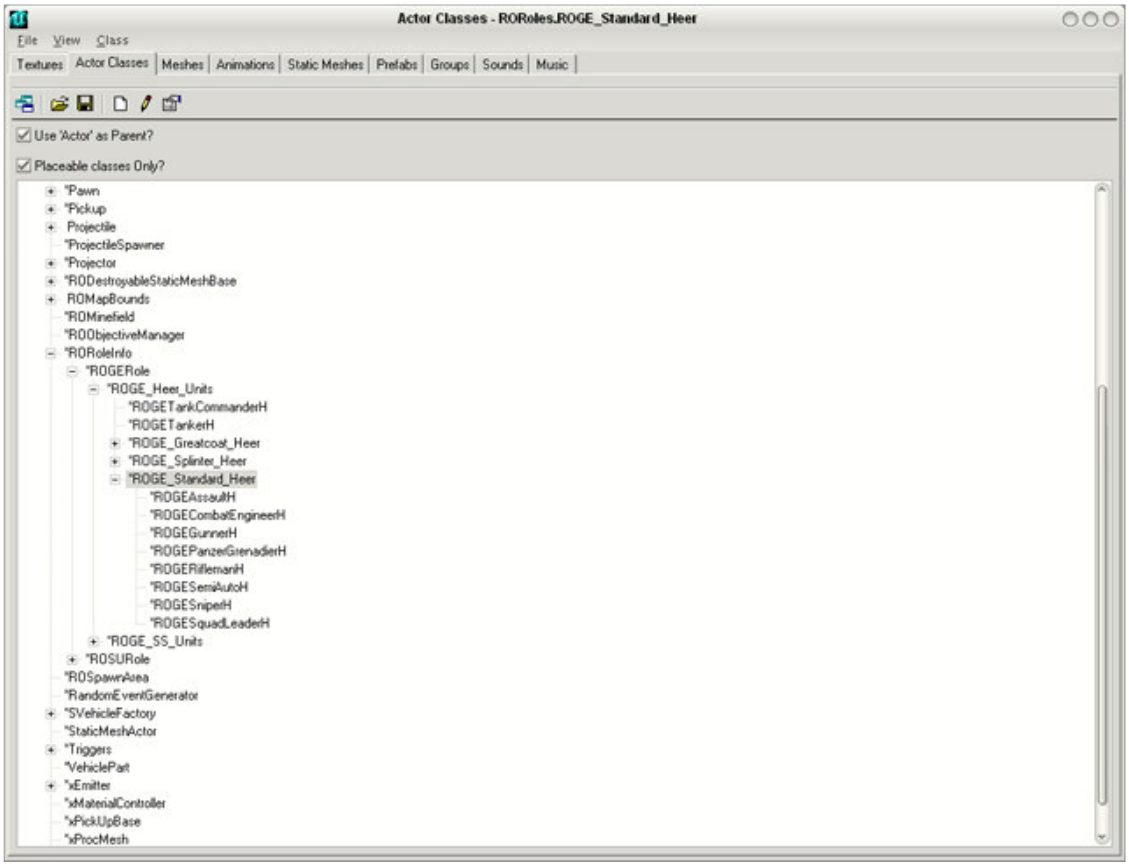
Roles also incorporate other concepts such as nationality and service branch. Of course, Germany's Wehrmacht and the Soviet Red Army are present but, in addition, other service branches such as Luftwaffe Field Divisions and the NKVD (Soviet security troops) may be represented.

Why bother with all of this? Well, as a level designer, you are free to configure the types of roles allowed in the level to suit the battle your map recreates. As an example, both sides quickly learned the value of the sub-machine gun in the ruins of Stalingrad. In a map based on that battle, you might want to allow more assault role slots than you would normally.

In order for a Red Orchestra: Ostfront '41-45 map to be playable, each side must have at least one role assigned.

Let's get started then. Open the editor and load our quick start map. Since roles are map actors we'll once again be working with the actor browser. Open the browser and search until you find the RORoleInfo actor. The plus sign beside the actor lets us know that RORoleInfo represents a group of related actors. Click on the plus sign. Open the ROGERole actor group, then the ROGE_Heer_Units group and finally the ROGE_Standard_Heer group.

After all that clicking you should see something like this:



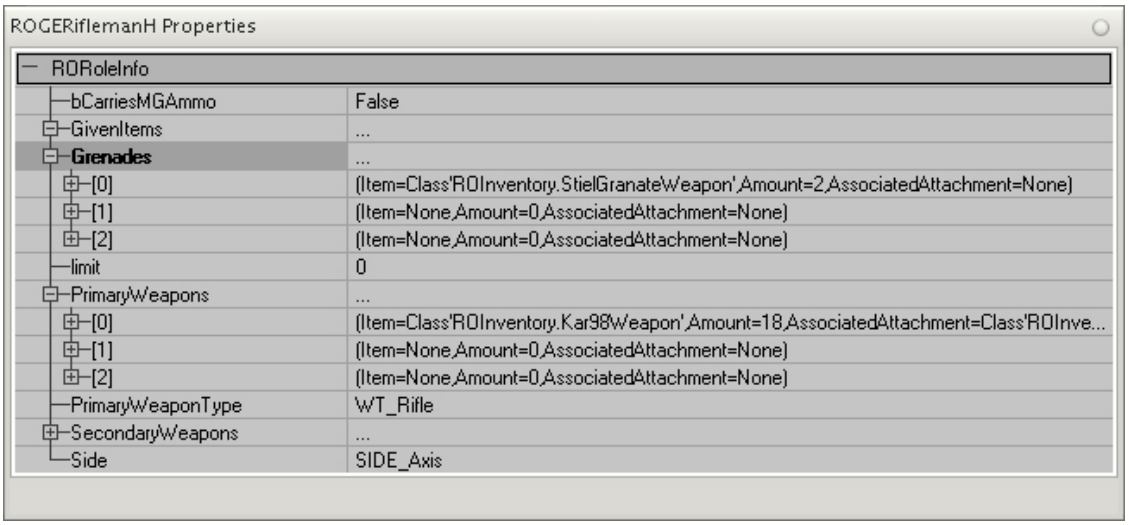
This gives you an idea of the organization of roles. We opened the German role group first. Under this was the "Heer" (or army) group of roles, which we opened second. Finally, roles included in the ROGE_Standard_Heer group are associated with standard army uniforms. Looking at the browser, you can see we could have also chosen uniforms with greatcoats and "splinter" uniforms (which refer to splinter units such as the Luftwaffe field divisions).

Since the rifleman is the most exciting and challenging role (and anyone who says differently just hasn't fully thought it through yet) we'll add that one to our map. Select the ROGERiflemanH role then close the browser.

We can now place the actor in the level. Role actors, like the ROLevelInfo actor, aren't position dependent so let's put it somewhere out of the way. In fact, why not put it right where we put ROLevelInfo? Remember, now that you've selected the role actor you can place it in the map by right clicking to bring up the context sensitive menu and selecting "Add ROGERiflemanH Here". It's best to do this in the rendered view pane just to be sure it is placed properly.

Once that's done, we'll need to add a Soviet role. Open the actor browser again and find the ROSURiflemanM35RKKA role. Place it in the level next to the German version.

Now, just for interests' sake, let's take a look at the role properties. Double click on either of the roles to open the properties dialog.



The most important property right now is "limit". This property is used to set the maximum number of players that can use the role during a game. Zero has a special meaning: it means that there is no limit on this particular role. That's good.

That's it. Rebuild the level and save it. You can now play the level in Red Orchestra: Ostfront '41-45. It won't be much more than a deathmatch level at the moment, but we'll do something about that in the next part of this tutorial. Right now, I'd suggest trying out your level.

1.7 Objectives

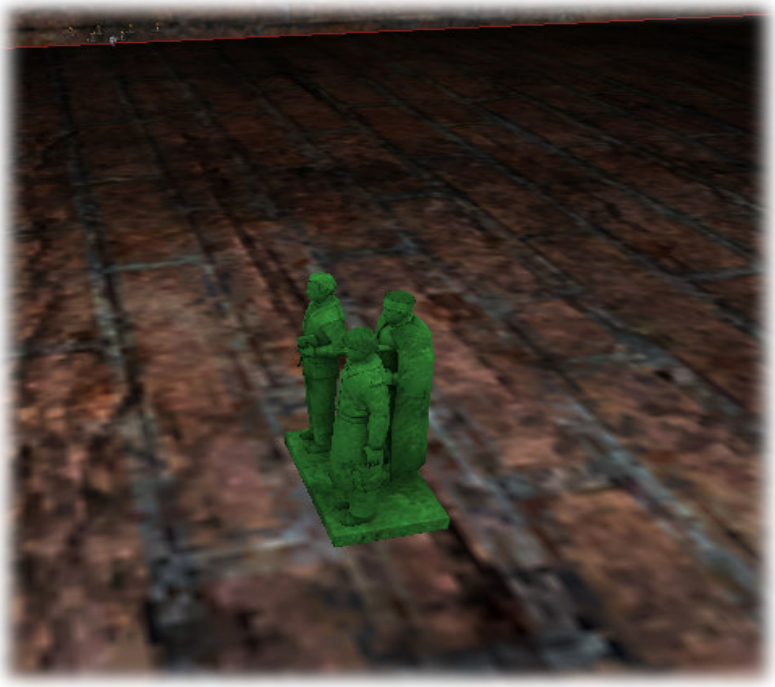
We now have to turn our level from its current state as a glorified World War 2 deathmatch level into something with more of a point to it. We'll do this by adding an objective.

Open the Red Orchestra level editor again and load the quick start map. We want to be somewhat fair about this so we'll put the objective in the middle of the map. Right now, the map is kind of empty, which makes it a bit difficult to actually find the objective. Let's change that by adding a static mesh to the map at the location of our objective.

What's a static mesh? Put simply it is a model created in an external program such as 3ds Max or Maya. Static meshes are normally used to decorate a map and will include everything from walls and buildings to books and grass.

Open the browser and select the static mesh tab. Hit the open package button and select the UrbanSM package. This loads a group of static meshes into the browser where you can view them in all their three dimensional glory. The first mesh in the UrbanSM package should be the 3manstatue, which looks like it would make a suitable landmark for an objective. Close the browser and move to the center of the map in the rendered view pane.

Right click on the desired position for the mesh to bring up the context sensitive menu. You should see an option to add the static mesh "UrbanSM.statues.3manstatue" to the level. Do it. You should see something that looks like this:



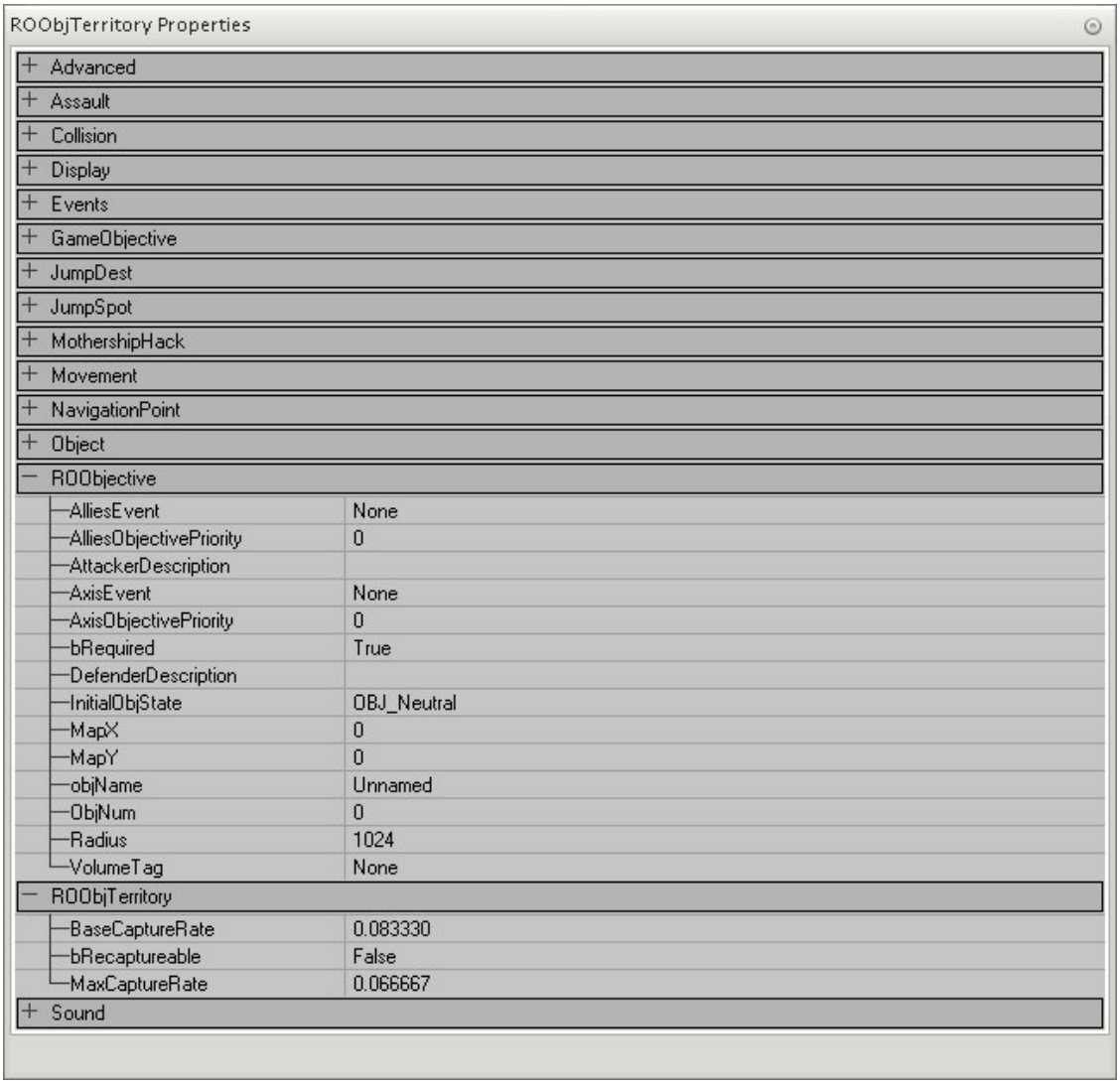
Now we can set up the objective. Open the browser again and switch to the actor tab. The objective actors are difficult to find, so I'll list the steps carefully. Look in:

1. NavigationPoint
2. JumpDest
3. JumpSpot
4. GameObjective
5. ROObjective

If you're wondering why the actor is located there it's because the objective has to be recognizable to bots.

We're going to add a territory objective, so select ROObjTerritory and close the browser. Objective actors are position sensitive so try to place it relatively close to the statues. If you don't remember, you can place the actor by right clicking to bring up the context sensitive menu then selecting the option that says "Add ROObjTerritory Here".

Notice that the ROObjective actor looks like a Red Orchestra icon. This is just to help you quickly recognize the actor when you're looking at a map. Double click on the actor to bring up its properties.



ROObjectiveTerritory is a little different from the other actors we've dealt with in that it has two sets of properties. The first set is shared by all members of the ROObjective family of actors, while the second set is specific to ROObjTerritory.

Most of the defaults here are fine for our purposes. We will change a couple of properties, though. First, set the InitialObjState property to OBJ_Allies, since the Germans are attacking. Then change the objName property to "The Statue" (you'll see this on the objectives screen now). Next, change the AttackerDescription to "Attack the Statue!" and the DefenderDescription to "Defend the Statue!" (this is an important statue, after all). Finally, set the VolumeTag property to "ObjectiveVolume". We'll talk about this in just a little while.

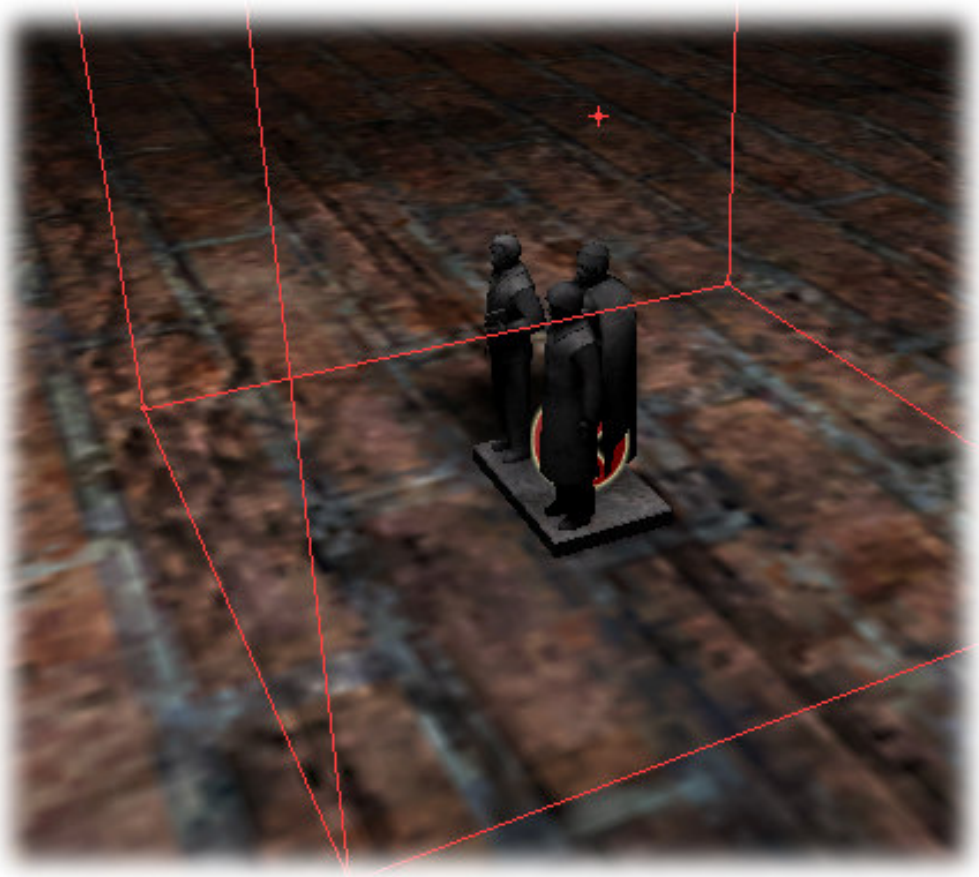
Before we leave this actor, we'll set one more property: the bRecapturable property in the ROObjTerritory group. Set this property to "False", since we don't want to give the Soviets the chance to recapture it.

Now that we've got our objective established, we need to define the area that it covers in the map. The area that is included in an objective is entirely up to you, the level designer. If you don't care about this too much you can always set the objective to include anything within a certain radius. If you do care, then you have to define a volume around the objective. That's what we'll be doing next.

A volume is a special map actor. It is created by defining a brush and modifying it until it is the size and shape you need. The volume then created much like a bsp.

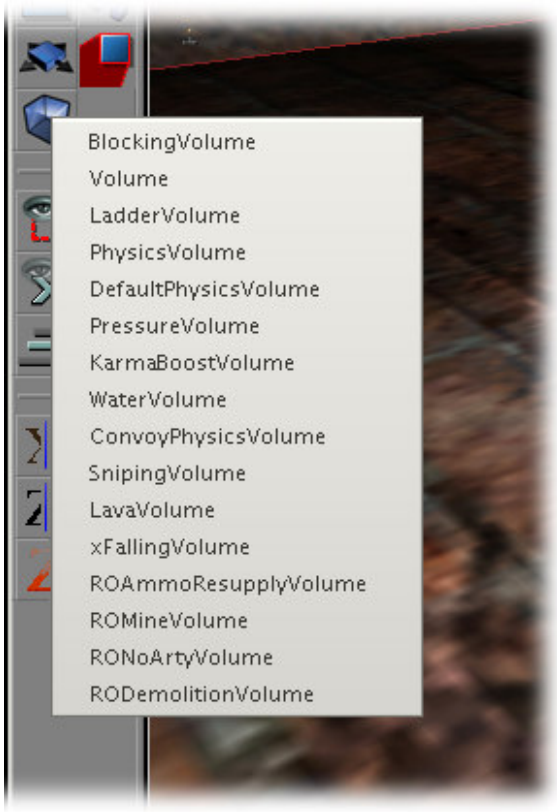
First, let's create our brush. For the purposes of this tutorial, we'll keep it simple and just define it as a cube. Right click on the cube icon on the tools palette. If you don't remember, it looks like the picture to the left.

Right clicking on the cube should bring up the cube properties dialog. Set the size of the cube to 512 units a side. That's enough for people to stand in but not so big that most of the map is included in the objective. When you've finished that, click on the Build button to create the brush. Then move the brush to the proper place so that the bottom of the brush rests on the bottom of the level and it completely encloses the statue objective (drag the box while holding down the control key). If you've done it properly, it should look something like this:



Now we're ready to create the volume.

The volumes are found on the same palette as the cube, just a bit further down. It looks a bit like a blue, transparent cube. Right click on it to bring up a list of volume types. That list will look like this:



We're interested just a plain volume, the second choice. Select it and then move the brush a bit. You should now see a pinkish cube where the brush used to be. That's your volume.

Now, it's a bit finicky, but double click on the volume. This will bring up its property dialog. Go to the Events property group and open that up. Set the Tag property to "ObjectiveVolume". We have just linked this volume to the ROObjTerritory actor we defined earlier.

That was a bit tricky, but we're done now. Rebuild the level and save it. If you play the level now, you should notice the new objective. As a German player, you will now be able to run up to the statues and attempt to capture it. When you're successful, the level will end.

Congratulations, this is now essentially a working Red Orchestra: Ostfront '41-45 map. There are still a few things we can do to make it more use friendly, and we'll look at those in the next sections of this tutorial.

1.8 Setting Up an Overhead Map

Even though our quick start level is not fully functional, there are still a few things we can do to make it even better. You will probably have noticed that all Red Orchestra:Ostfront '41-45 maps sport an overhead map to help orient the players during battle. We can easily add one of these to our map now.

In order to do this, we'll need two things. The first is some graphic to use as an overhead map. For this purpose I've provided a simply screen capture of the level. It's quite dark and there's not a lot of detail but it will serve our purpose nicely. You'll find the screen capture in a file called QuickStartOverheadMap.bmp accompanying this tutorial.

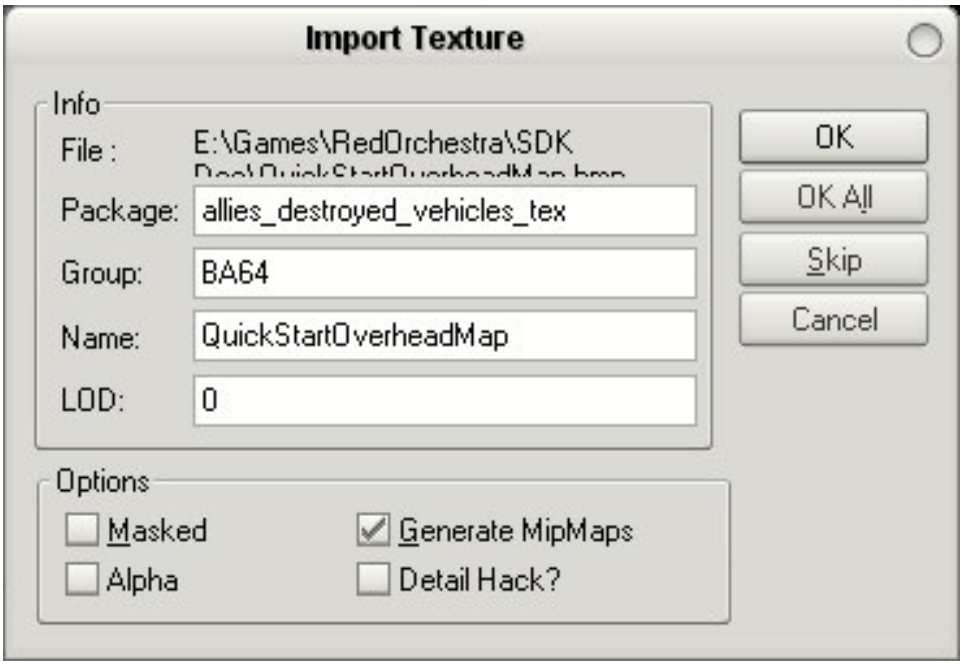
A map's not enough on it's own, however. We have to place the map properly with respect to the level itself. This is done in Red Orchestra: Ostfront '41-45 via the ROMapBounds actors. There are two actors we must place: ROMapBoundsNE and ROMapBoundsSW. As you can logically deduce, the first goes in the extreme northeast corner of your map while the second is placed in the extreme southwest corner.

Start by firing up the Red Orchestra level editor and opening the quick start map. Before we begin, we must first decide where north is. Since I took the overhead screen capture facing the same way as the statues that make up our objective, we'll arbitrarily say that direction is north. So, if you look down on the map with the top being the side the statues face, then the ROMapBoundsNE actor goes in the top right corner and ROMapBoundsSW in the bottom left. Add these actors in the rendered view pane just like the all the others. When you're done, you should see something like this:

Note: You will have to place the map bounds actors slightly differently when you want to use artillery, but we'll leave that for a future tutorial. This setup will work when artillery is not in use.

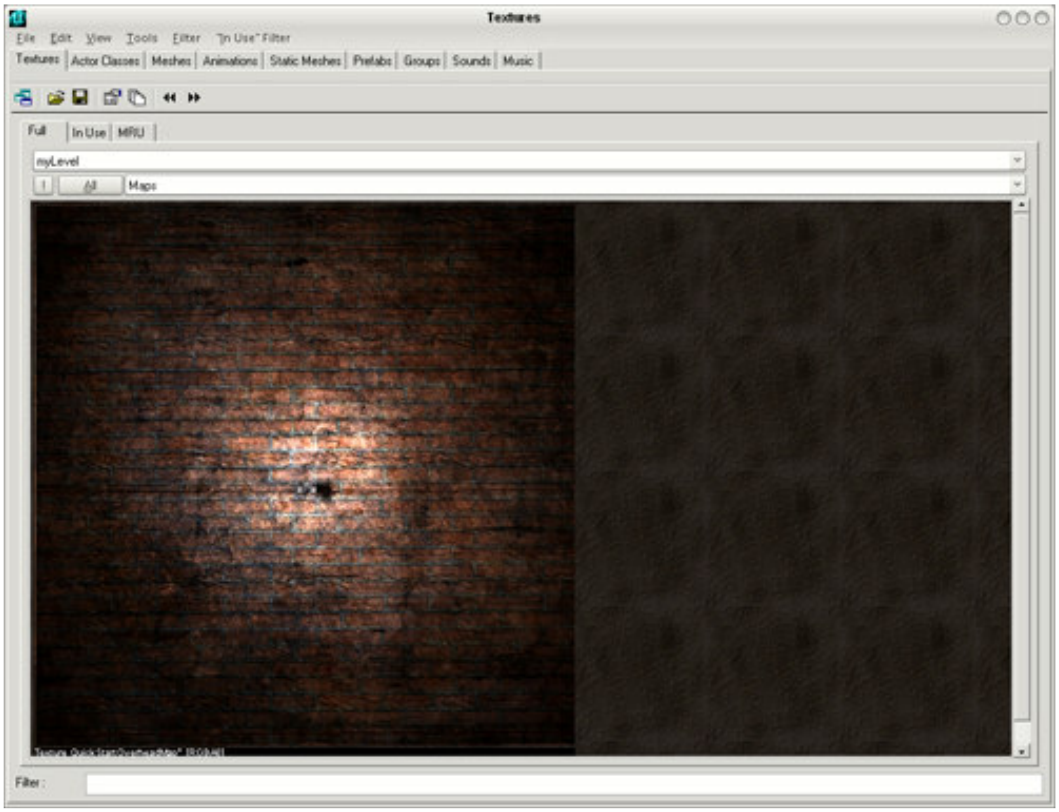
Our next step is to introduce our overhead map to our level. To do this, we'll have to import the graphics into the editor. This is something that level designers have to do quite frequently, so it's a good thing to know.

Open the browser and go to the texture pane. Under the File menu, find the entry that says "Import…" and select it. Now find the QuickStartOverheadMap.bmp file and open it. You'll be presented with a new dialog, like this one:



We're now going to tell the editor where to file our overhead map. Set the Package field to "myLevel". The myLevel package is a special location commonly used for most custom map graphics and we'll use it now. Set the group field to "Maps". Finally, uncheck the "Generate MipMaps" checkbox. Now hit "OK".

If everything went correctly (and we know that nothing ever goes wrong in a tutorial), you should now see the overhead map as a texture in the browser.

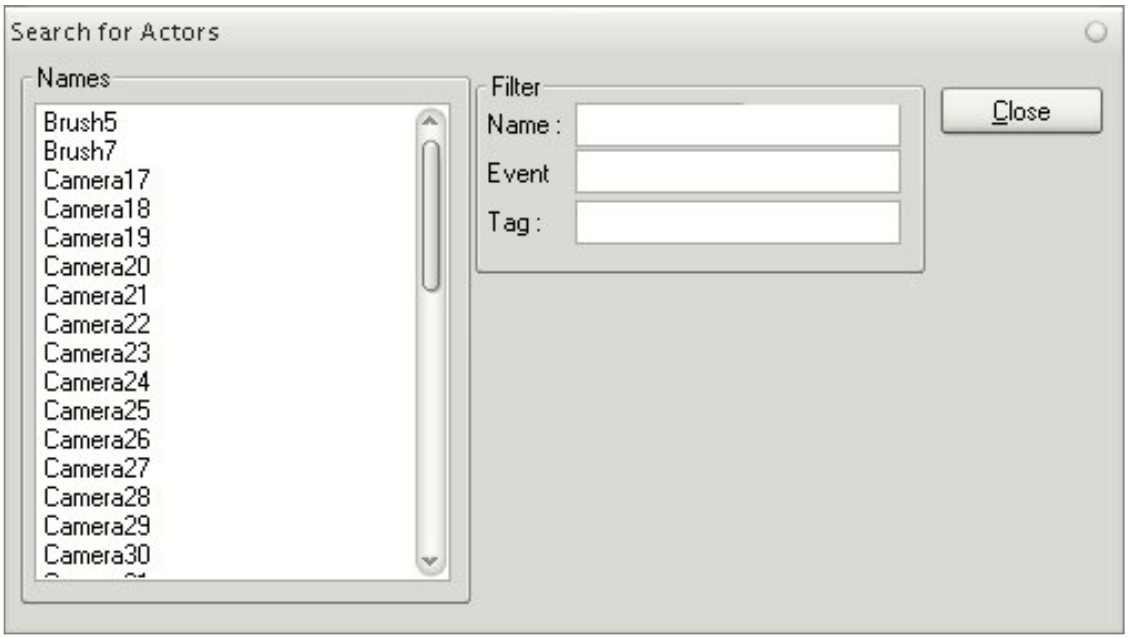


Before we move on, there's just one more small task to perform. Right now our overhead map is stored in RGB8 format (you can see this for yourself by looking at the texture details at the very bottom of each texture in the browser). We don't need the map in this format, which uses quite a bit of space, so we'll compress it. Right click on the texture to bring up a context sensitive menu. Select compress and then DXT1.

I won't go into the various compression options available here at this time. Suffice it to say that the overhead map texture will now take up less space and still be quite usable.

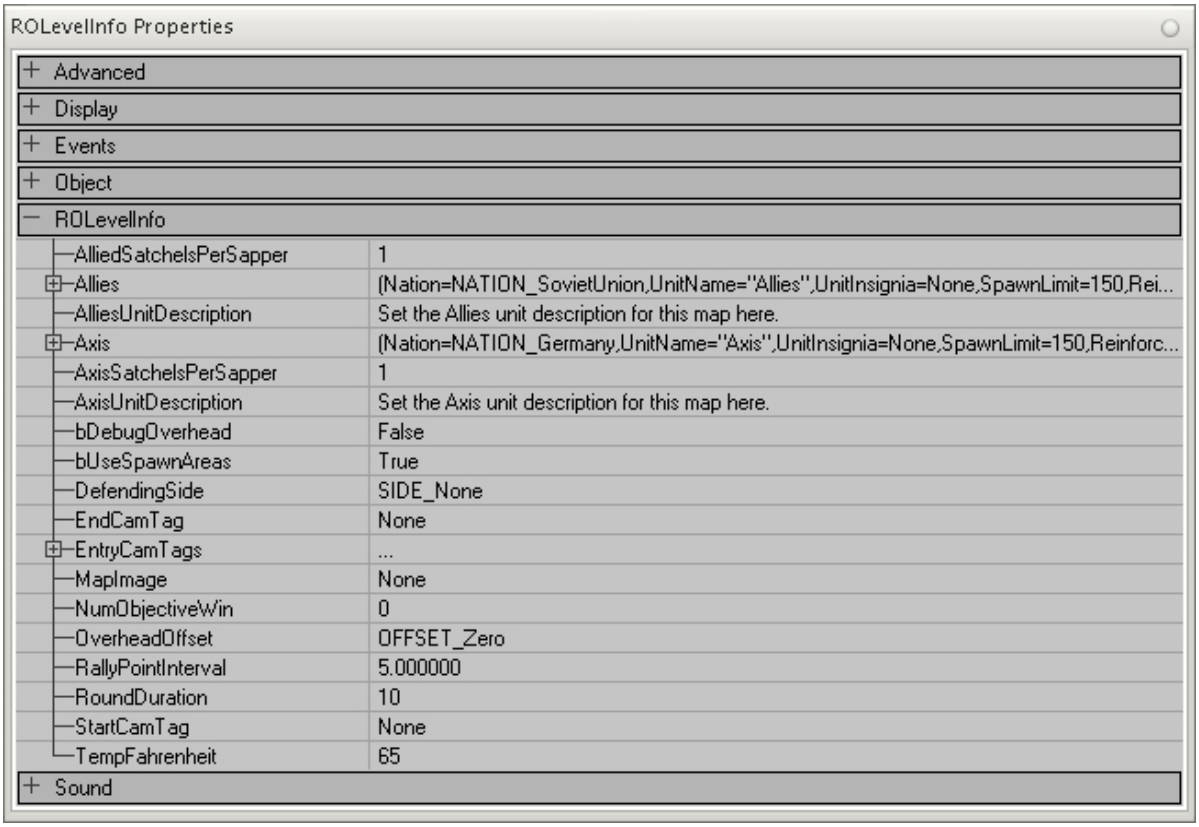
We're now almost done. All that remains is to actually insert the map into the level. We do that with our old friend the ROLevelInfo actor. You may have forgotten where you put this actor, so I'll now show you a quick way to track down errant map objects.

Up on the top toolbar of the editor you will see a button with an icon that looks like a pair of binoculars. This is the search tool. Press the button and you should see a dialog that looks like this:

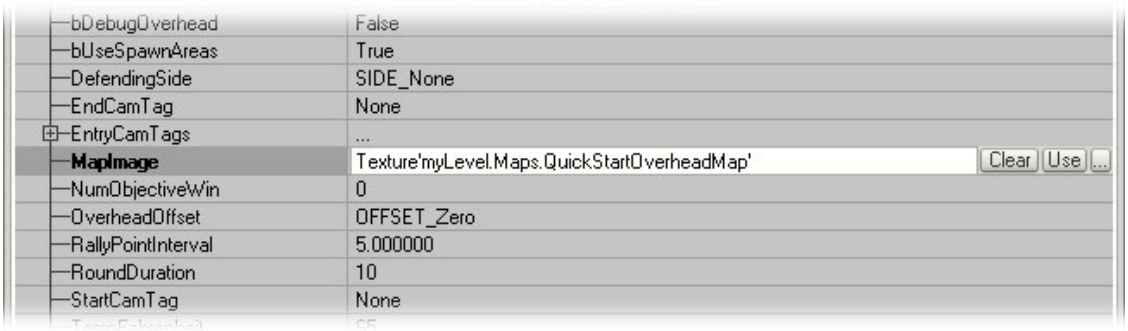


Type "ROLevelInfo" in the Name field. Double click on the name that remains in the left hand scroll box then hit Close. The editor will now find that actor and set the focus to it. Very useful.

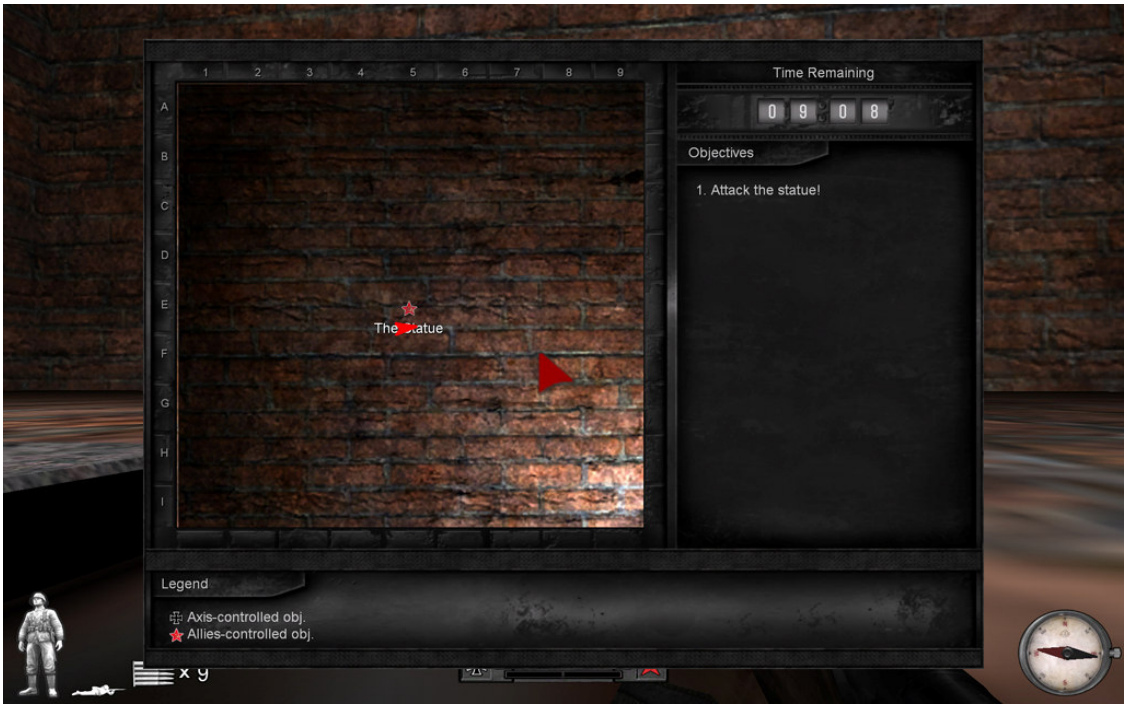
Double click on the ROLevelInfo actor to open its property dialog then find and open it's properties. Scroll down until you find the property "MapImage".



Our map image texture should still be selected from when we compressed it, so click on the MapImage property then click on "Use". If some other texture besides the overhead map appears in this property, simply go back to the texture browser and select the overhead map texture then try again. When you're done the property should look like this:



It's time to rebuild and save your level. Now, when playing, you will be able to invoke the overhead map and see where you are in the level. Your map should look something like this image below:



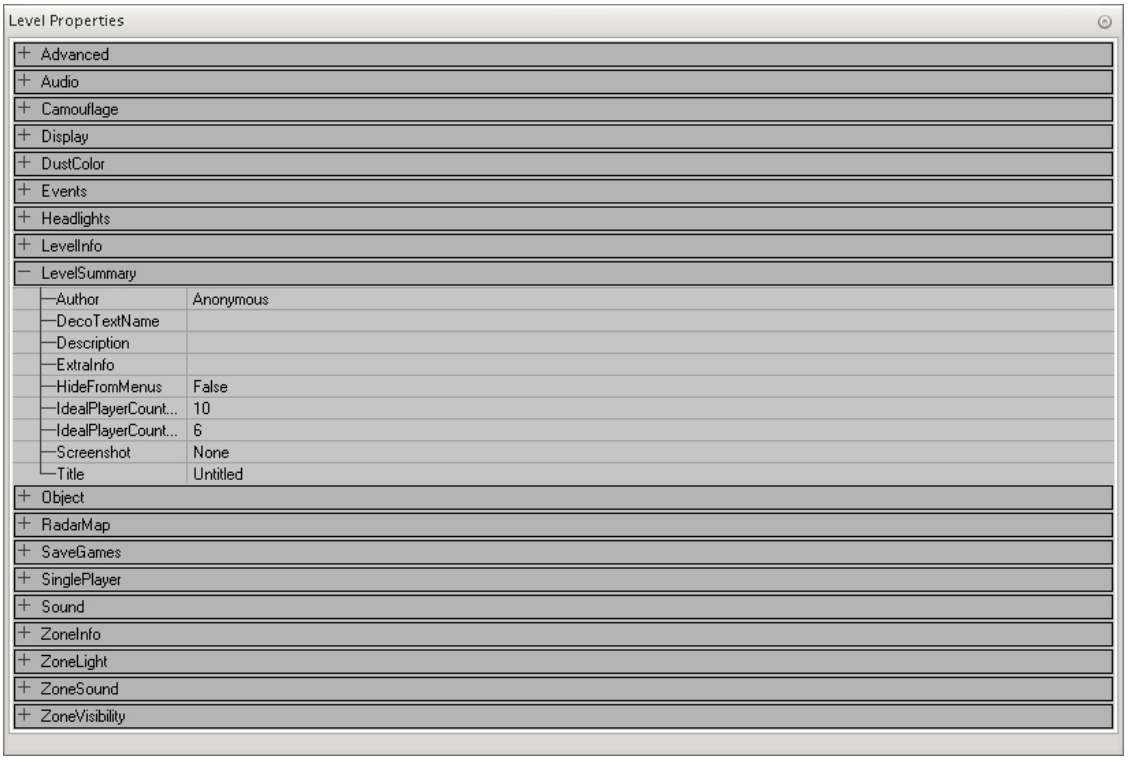
So, that's a bit of work but as you can see, the results are worth it. We're almost done with the quick start level at this point. Just a few more, final touch ups and we're finished. We'll tidy up all the loose ends in the next section.

1.9 Cleaning Up

It's now time to finish off this quick start level. One thing you've probably noticed when playing with your level is that it doesn't have a nice introduction like the ones that were provided with Red Orchestra: Ostfront '41-45. Well, let's fix that right now.

Each level in Red Orchestra: Ostfront '41-45 has a level summary that provides a bit of background information. This can be quite important in Red Orchestra, as there's almost always some interesting history associated with each level. This summary is set in the level properties. The level properties aren't set in any map actor, but instead are configured in the editor itself.

Start the Red Orchestra level editor and open the quick start level once more. This time, look at the View menu and find the option "Level Properties". Select this and the level properties dialog will appear. Open the LevelSummary group so you see the following:



First, set the Author property to your name. This lets players know whom they have to thank for the level they're playing (this may or may not be a good thing). You should also always set the Title property to the name of the level (in this case RO-QuickStart).

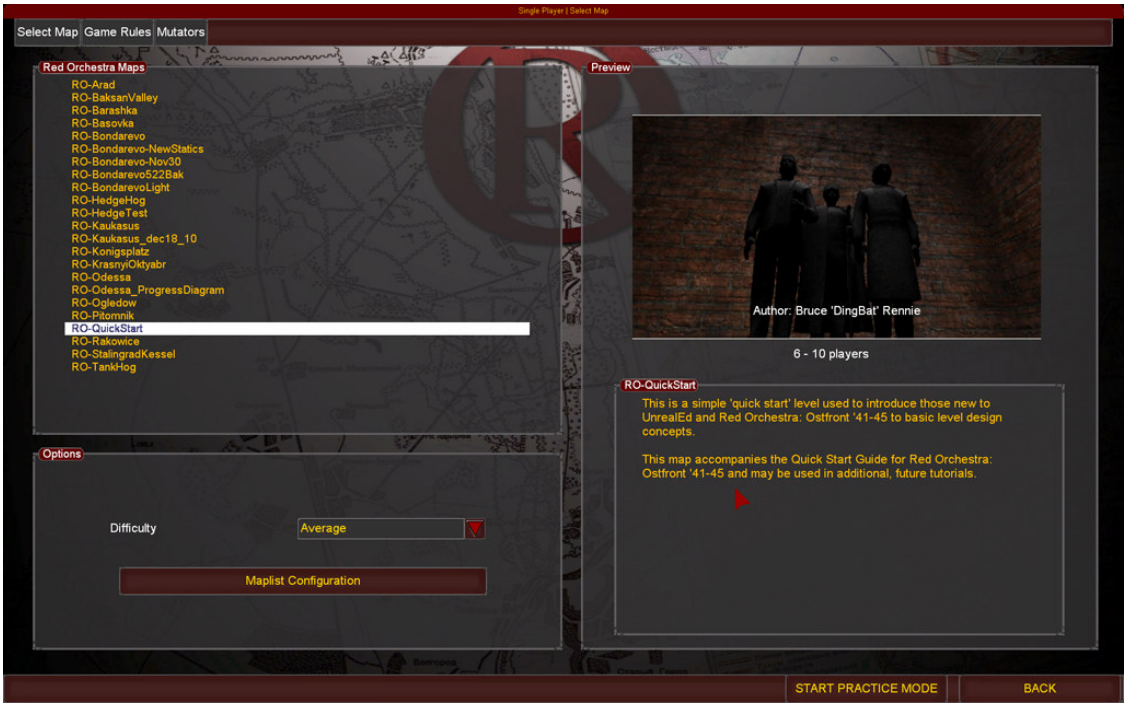
The next most important property is the Description. Using this property you can add a nice piece of background information for the level. The only trick you need to bear in mind is that line breaks in the text are entered as "|". For example: "This is line 1.|This is line 2.". Enter anything you want in the description now.

Level designers can also set the ideal player counts for their level using the IdealPlayerCountMax and IdealPlayerCountMin properties.

Finally, we can also set a teaser screen shot while will be displayed in the level selection screen. For best results the graphic used here should be 512x256 in size. I've provided such a screen shot here so you don't have to generate one yourself. It can be found in the file QuickStartPreview.bmp and should accompany this tutorial.

Before we can add the preview screen shot, we have to import it into the editor, just as we did with the overhead map. You can refer back to the previous section if you've forgotten how to do this (don't forget to compress the image). Once that's done we can then add the texture to the Screenshot property.

Rebuild and save your level. Now, when looking at the level in the selection screen, you should see something that looks like this:



1.10 Conclusion: Where to Go From Here

Congratulations. You have successfully created and configured a working Red Orchestra: Ostfront '41-45 level. Hopefully, this tutorial has simply whetted your appetite for level development and you will now move on to more ambitious projects.

You probably realize already that it's a long step from this sample map to a level like Konigsplatz. I'll take it for granted that this is your ultimate goal, however. If so, where should you go from here?

There are 3 things you'll have to consider when starting your first actual Red Orchestra: Ostfront '41-45 level:

1. Setting
2. Gameplay
3. Construction

Red Orchestra players tend to value realism. Maps that exhibit gross historical inaccuracies aren't likely to be successful, even if they are good maps. There are numerous popular histories of the fighting on the eastern front. Becoming familiar with these isn't a bad way to start developing a level.

Gameplay is a critical factor in any level, for any game. In order to design a map that plays well you have to understand the subtleties of the game itself. The best way to gain that knowledge is to play. A lot. But when you do play, play as a level designer. What works about the level? What doesn't? What could be improved? What new twists could be added to make the experience even more enjoyable? It may not feel like you're working on a level when you play, but I've found that every game session can teach you something about level design.

Finally, you have to be able to turn your vision into reality through the Red Orchestra level editor. You have to know what's possible and what isn't and how to make what is possible actually work. Look for more Red Orchestra: Ostfront '41-45 level tutorials to help you there. There are also a large number of excellent tutorials on general UnrealEd concepts out there to help you. Find them, read them, and then use them.

By the way, I'd recommend keeping this quick start level around. It doesn't look like much, but it's ideal for testing out new ideas. I may also use the map in future tutorials that examine more advanced level design techniques.

That about wraps things up for this tutorial. I hope you enjoyed it and found it useful. Good luck.

1.11 Converting old MOD maps to Retail

Converting maps from the MOD to retail will take quite some work. None of the static meshes, textures or even Red Orchestra Specific Actors like the Spawnareas, Objectives or Managers are compatible between the MOD and Retail version of the game. This is simply because so much has been updated, replaced and improved on in the retail version of the game.

Especially leaving the gameplay actors and special effects like particle emitters and coronas in the map will crash your map in the editor, and malfunction the map in the retail game.

Leaving the static meshes and textures that are used will mostly give you load errors when you open your converted map in the RO editor. The editor will tell you it can't find the resources for all the old stuff and will tell you that on re-saving the map, all instances will be set to NULL, which means all instances will be deleted from the map database. This will leave you with a basic map containing the terrain without textures, BSP without textures and no static meshes.

1.11.1 Porting procedure from Mod to Retail

The procedure to safely port an old map from the MOD version to retail is as follows:

1. Open the MOD map in the MOD editor. An obvious comment is to take a backup copy of your work before you start, as well as at various stages through this process.
2. Delete ALL non standard UED gameplay actors. You can leave UED standard actors listed below in your map:
    a. PlayerStarts
    b. Zoneinfo's
    c. PathNodes
    d. AssaultPath
    e. RoadPath nodes
    f. DefensePoints (UnrealScriptedSequence)
    g. Snipervolumes
3. Be careful with leaving Scripted sequences, emitters etc in your map. Most likely they can have links to old sounds, old textures and/or old static meshes embedded in their scripts. When these are not removed properly from the scripts, the map will keep on crashing or fail to load in Retail and be tricky in debugging! To be safe it is best to remove all these as well.
4. You can leave all UED BSP and volume brushes in the map. You will however need to delete any RO specific volume from your map, except the RODemolitionVolume. That one is still compatible with Retail.
5. Delete all particle emitters (these mostly will use UT or MOD textures or meshes which are no longer in the game).
6. To be safe we recommend deleting all static meshes. Only the static meshes that are in the Retail version are properly owned by Tripwire and can be used in your custom or ported maps. All other MOD-specific material is most likely owned by someone else and you may be violating copyrights/ownership rights when you keep them in the map.
7. You can leave all textures on the BSP, but they will be removed (set to NULL) when you open your ported map into the new editor. The same ownership

rule described for the static meshes is true for the textures; if they are not in the retail version, Tripwire does not own the rights to them.

8. With all that done, save your map as a <RO-MapName>.ROM file, which is the basic naming standard for Red Orchestra Ostfront map files. Note: Both the prefix RO- and the suffix .ROM are mandatory for your map to load in the retail game.
9. If you have saved the map in the MOD's Maps directory, copy the map to the Retail Maps directory manually, so you can open it up from the retail editor.
10. Open your RO retail editor
11. Open your newly saved conversion map
12. One of two things will happen:
    a. The map will open fine, and has some resource errors. All you will have to do now is resave the map (preferably under a different name). This will effectively automate the removal of all unknown resources in your map. When this is done, you can start adding the basic gameplay actors like the ROLevelInfo, RORoles (the soldier classes) to see if the map will load in the retail version of the game.
    b. The editor crashes when you try to load the map. This will be because you still have links to old resources in your map. This can range from RO gameplay actors to an obscure texture used in a light as a corona. You will have to go back to the last saved version in the MOD editor and find that offending resource instance. This will be trial and error. Keep deleting old stuff until the map loads in the retail editor.

What you will end up with is a bare map stripped of all the gameplay actors, textures and static meshes. You can now start filling in the blanks and start using the new artwork that comes with the editor.

1.12 UnrealEd is a Harsh Mistress

UnrealEd is a complicated piece of software and it has its fair share of issues. With experience, you will find you can avoid most of UnrealEd's sharper edges and work relatively trouble free. In the meantime, here are a few tips that may help smooth out the ride.

1) When I open UnrealEd for the first time, all I see is a solid white window.

This is a pretty common experience and, thankfully, it's easily fixed. Go to the View menu then select Viewports (second last option) and then Configure. You'll be presented with a dialog that allows you to set the arrangement of views in the editor. Select any configuration and hit OK and you should be fine.

2) I can't select any surface.

If you are encountering this problem it is likely you have anti-aliasing set on your video card. Turn it off and everything should work properly.

3) I can't see that red brush in any view pane.

It's possible you accidentally hit the "B" button, which toggles the display of the builder brush.

4) That red brush is all skewed and doesn't align to the grid lines in the view panes.

That's easily fixed. Right click on the brush to bring up the context sensitive menu. Select "Reset All" to reset the brush's orientation.

If this list isn't enough, there are many other references on the Internet that may help.

Part 2: Spawn Areas Tutorial

2.1 Introduction

It might surprise some readers when I say that I consider the management of spawn areas to be one of the most important factors affecting game play in a Red Orchestra: Ostfront '41-45 level. Still, when you think about it, it makes sense. Spawn areas are where players enter the game so it's naturally going to have an effect on play. Sure, a lot of other things affect game play as well but if you mess up the spawn areas it's going to be very difficult to get the kind of results you want.

This tutorial will help you get the most out of your spawn areas. I'll start out with a simple example, just to get our feet wet. After that we'll move on to cover how to control multiple spawn areas. Finally, I'll show you a little trick I discovered while working on my first level that will help you make sure your levels start out right.

Spawn areas and objectives are closely related in Red Orchestra: Ostfront '41-45. This tutorial assumes you have a basic understanding of the Red Orchestra level editor and that you understand how to create and configure an objective. If not, I'd recommend first reading the Advanced Objectives Tutorial.

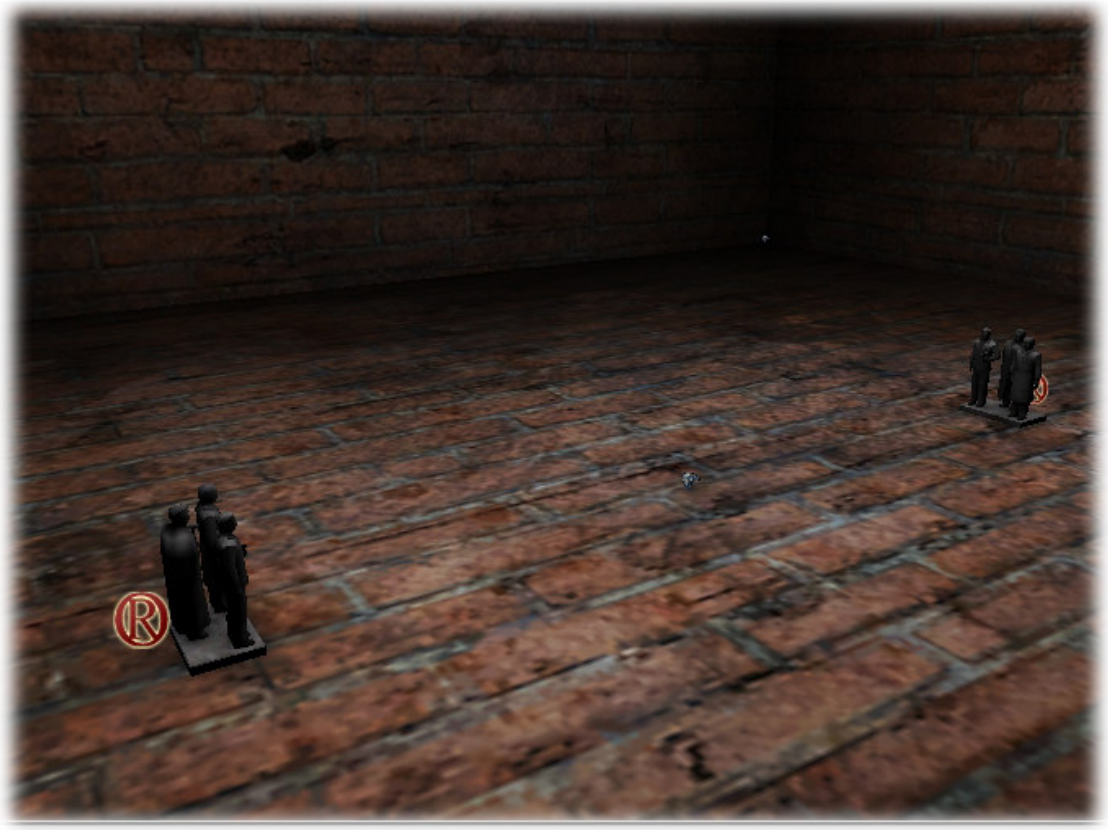
Normally, I'd base this tutorial on the sample level created in the Quick Start guide. However, since we're going to have to be creating a lot of objectives in order to establish the situations we want, I'll provide starting levels for you to work with. You should find these levels attached to this document.

As a warmup, let's review how to establish a basic spawn area.

2.2 Simple Spawn Areas

Before we dive into fancy spawn area setups, let's take a minute to go over a more basic configuration.

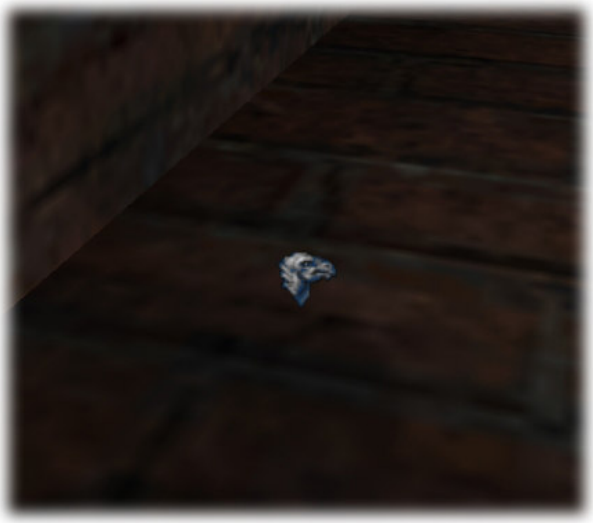
Start up the Red Orchestra level editor and open the map RO-AdvancedSpawnAreasP1, which should accompany this tutorial. This is a simple map which is very similar to the one constructed in the Quick Start Guide.



You may have noticed that the map has no spawn areas currently. As a bare minimum, every Red Orchestra: Ostfront '41-45 will need at least two ROSpawnArea actors: one for the Germans and one for the Soviets. Additionally, each spawn area will be accompanied by a group of PlayerStart actors. The spawn areas control which set of player starts are in use at any one time while the player starts themselves control the exact location a player will enter the game.

So let's begin adding spawn areas to our map. We'll place them on either side of the map, on the same line as the objectives. It doesn't matter which one we start with, so I'll start with the German spawn.

Open the actor browser and find the ROSpawnArea actor. Select it then close the browser. You can now place the actor in the map by right clicking on the desired location in the rendered view pane and selecting "Add ROSpawnArea Here" from the context sensitive dialog. You should see a new actor appear in your map that looks something like a horse's head.



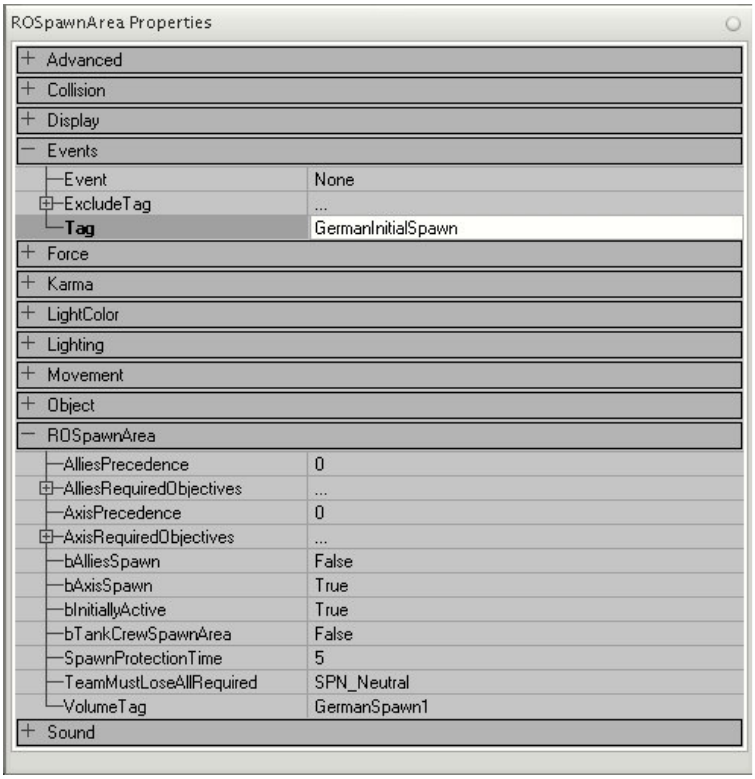
Double click on the actor to open its property dialog. Open the ROSpawnArea property group and set the following properties:

- bAxisSpawn: True
- SpawnProtectionTime: 5
- VolumeTag: GermanSpawn1

So what did we do here? We just told the spawn area that it will be used to spawn players on the axis team. The players will be immune from damage for 5 seconds after spawning (unless they, themselves, fire). The area of protection will be defined by a volume called "GermanSpawn1".

We're not quite done yet, however. The most important property of a spawn area isn't even in the ROSpawnArea group. Open the Events group and set the Tag property to "GermanInitialSpawn".

When you're done, the property dialog should look like this:



We're only half done with the German spawn at this point. Now we need to add the actual player starts. Right click on a point near the ROSpawnActor and select "Add Player Start Here" from the pop up menu. You will see something that looks like a joystick appear next to the spawn area actor. This is the actual location at which a player will enter the game.

You may notice a red arrow attached to the player start. This indicates the direction the player will face when they spawn. It's no good if the player start is facing a wall since no one wants to spawn with their back to the enemy.

You can rotate this arrow until it faces the correct way by selecting the player start in the top view pane and dragging the right mouse button while holding down the control key. If everything is done properly, the player start should look like the picture to the left.

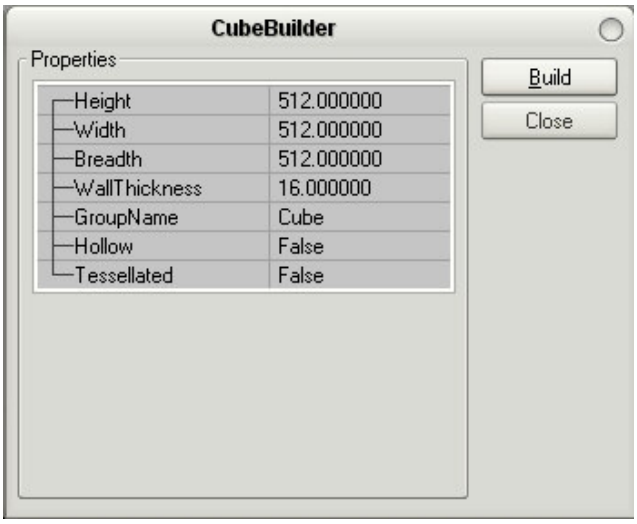
Finally, double click on the player start to open its properties dialog. Open the Event property group and set the Tag property to "GermanInitialSpawn". That tells this player start that it is associated with the spawn area actor we just configured.

So far, so good, but we're going to need more than one player start. Normally, in a Red Orchestra: Ostfront '41-45 level the maximum number of players on a team is 16 so you'd need at least that many player starts for each spawn area. I'd recommend a few more though, just in case there are collisions or someone is hanging around areas they shouldn't. Eighteen or twenty player starts should be fine.

I'm not going to bother adding 19 more player starts, since we're only using this map for testing. However, I will show you how to quickly copy a map actor. Select the player start then right click on it to bring up the context sensitive dialog. Select the "Duplicate" command from the list. You will see a new player start appear very close to the first one. Too close, in fact. Move the new player start to a better position by dragging it while holding down the left mouse button and the control key. Make five new copies of the first player start this way. When you're done you should see something like this:

Since we don't want any sneaky Soviets killing the German soldiers when they're spawning, we need to set up a protection zone. You'll remember when we configured the ROSpawnArea actor we entered a 5 second spawn protection time and indicated we were going to define the spawn zone using a volume with the tag "GermanSpawn1". Now is a good time to set up that volume.

In order to create a volume we first have to define a brush of the proper size. We'll keep things simple and define our spawn protection zone as a simple cube 512 UnrealEd units on a side. Right click on the cube builder tool on the left hand toolbar (surprisingly, it looks just like a cube). You should see a dialog that looks like this:



Set the dimensions of the cube to 512 a side and click the "Build" button. You will see a red cube, known as the "builder brush" appears in the editor. We now have to place the cube where we want it. You can move the cube around by dragging it while holding down the left mouse button and the shift key. Make sure the cube encloses the spawn area and the player starts and rests on the floor of the level (it's actually ok if the cube extends into the bottom of the level). Now we can build the volume we need.

Find the button on the left hand toolbar that looks like a purple cube and click on it. You'll see a pop up dialog that looks like this:

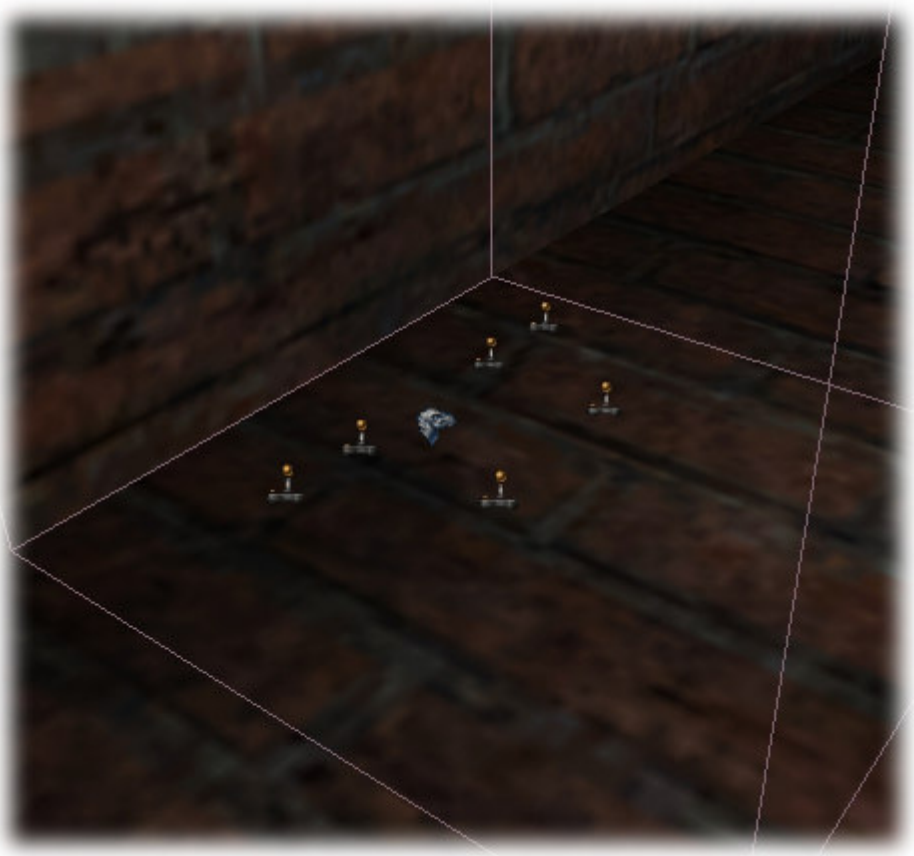
Select a plain Volume from the list. Now, if you move the red builder brush you will see a pinkish brush remains behind. That's our new volume.

Double click on the volume to open its property dialog. Open the Events properties and set the Tag property to the value we used earlier in the spawn actor: GermanSpawn1. Close the property dialog.

We have now configured the German spawn for our level. Unfortunately, I think we'll find the level rather dull with only a half-dozen German soldiers hanging around. We're going to have to create another spawn for the Soviets as well.

I'm not going to cover the steps to create the Soviet spawn but leave that as an exercise for you instead. It's exactly the same as the work we did for the German spawn except we'll need to change the tags we use in the player starts and volume.

When you're finished creating the Soviet spawn you're level should look like this:



Now is a good time to build and save your level. The level is actually playable at this point, so you can take a trip around the block, so to speak, if you want. In the next section we'll make things more interesting by adding a second spawn for the Germans.

2.3 Advancing Spawn Areas

If you're modelling a battle in which one side is expected to advance, then, if you stick with one spawn area, the players on that side are going to find their commute getting longer and longer. In most games of this nature fighting is fun, walking to the fight is not.

For example, in our test map, you probably noticed that we have two objectives. Even on this simple map, this presents a bit of a challenge for game play balancing. The Soviets will have a hard time defending the first objective from the Germans, while the Germans will have an equally difficult task taking the final one.

What we really need is spawn areas that advance with the action. Luckily, we can do exactly this in Red Orchestra: Ostfront '41-45.

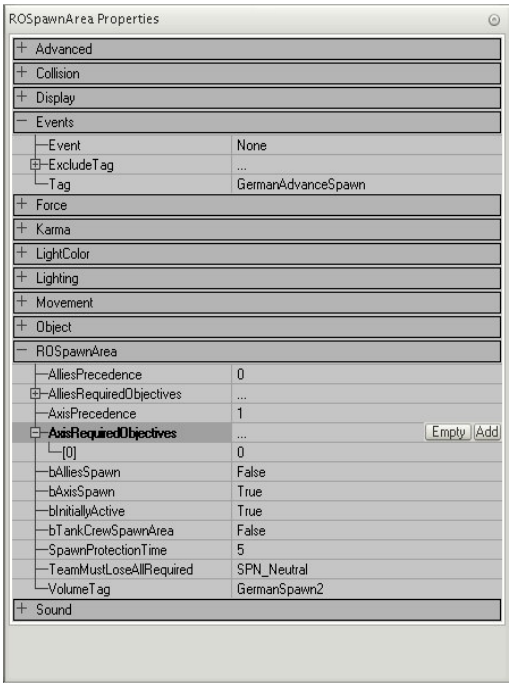
Start up UnrealEd and re-load the test map you completed in the first section. We'll try something simple first. We'll leave the Soviets as they are, but we'll create a new spawn area for the Germans that's closer to the second objective. At least then the fight for the final objective will be equal.

The first step is to place the new ROSpawnArea actor. Open the actor browser, find the actor and place it next to the first objective statue by right clicking to open the context sensitive menu and selecting "Add ROSpawnArea Here".

Double click on the new spawn area actor to open its properties dialog. Set the following properties:

- AxisPrecedence: 1
- AxisRequiredObjectives: 0
- bAxisSpawn: True
- bInitiallyActive: True
- SpawnProtectionTime: 5
- VolumeTag: GermanSpawn2

Remember, we have to set the Tag for this spawn area too. Open the Events properties and set the Tag field to "GermanAdvanceSpawn".

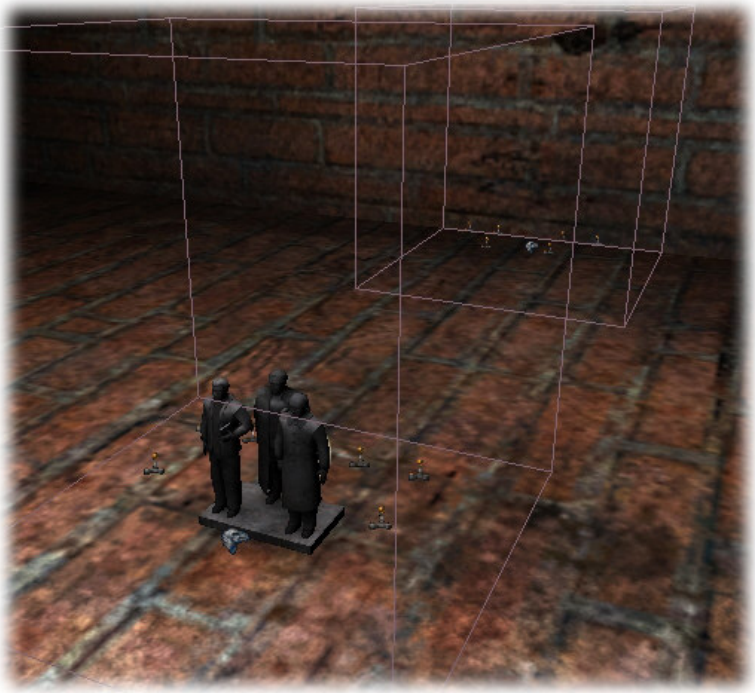


Let's go over what we've done here. Now that we have two spawn areas, we need to tell the game which one to use in which conditions. We've set the precedence of this spawn area to a value higher than the original spawn so that, if both spawns are active, the game will use this one. In AxisRequiredObjectives, we established the conditions under which this spawn area will activate. If the Germans capture the objective whose ObjNum property is 0, then this spawn area will activate.

Now we can add the player starts for the spawn. Again, you can place PlayerStart actors by simply right clicking in the appropriate location and selecting "Add PlayerStart Here".

Go ahead and place a half-dozen player starts around this new spawn. In fact, you can go ahead and add the spawn protection volume as well. You can refer back to the previous section for detailed instructions if you've forgotten. Just remember to make sure the tags of all the associated actors match up.

When you're finished, your level should look like this:



Rebuild and save you level now. Start Red Orchestra: Ostfront ’41-45 and play the level as the Germans. To test out our new advancing spawn you’ll have to capture the first objective and then “die” (so you can see for yourself that players will spawn at the new area). The easiest way to do this is to type “suicide” in the console. Go try it now.

You can easily establish exactly the same kind of setup on the Soviet side as well. The only difference is that we want the first spawn area to be the one closest to the first objective. The German spawn areas will seem to advance and the Soviet ones will seem to fall back. With this kind of spawn area arrangement, players will always spawn close to the action.

2.4 Complex Advancing Spawn Areas

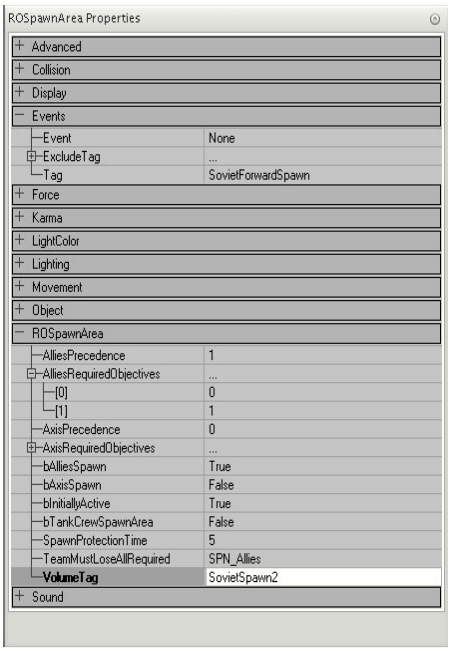
What if one team is defending multiple objectives simultaneously? It can get a little hairy trying to manage the spawns for that kind of situation. Fortunately, there's an easy way to manage this in Red Orchestra: Ostfront '41-45.

Fire up UnrealEd once again and this time open the map RO-AdvanceSpawnAreasPt2. A quick look will show you that I've set this map up so there are three objectives in this map: two up front and one to the rear. Our game plan is to have the Soviets spawn at the rear objective until both forward objectives are captured, and then fall back to the existing, final spawn area. We won't change the German spawn for this exercise.

Place a ROSpawnArea actor at the rear statue and set the following properties:

- AlliedPrecedence: 1
- AlliedRequiredObjectives: 0, 1
- bAlliesSpawn: True
- bInitiallyActive: True
- SpawnProtectionTime: 5
- TeamMustLoseAllRequired: SPN_Allies
- VolumeTag: SovietSpawn2

And then set the Tag property in the Events group to "SovietForwardSpawn".



Since we now have two spawns and we want the game to use this one if it can, we set the AlliesPrecedence property to a higher value than the rear spawn. We want this spawn to be used only if the Soviets continue to hold either of the objectives numbered 0 and 1, so we've entered those numbers in AlliesRequiredObjectives.

This raises an interesting possibility. We could just have easily decided that the spawn area should only be used if the Soviets hold BOTH objectives. How could we set that up? That's where the TeamMustLoseAllRequired property comes into play.

That property tells the game that the Soviets have to lose both of the required objectives in order for the spawn area to become unusable. If we had wanted the spawn to be disabled when either objective was taken, we would simply have left this property alone.

We'll need to finish up the housekeeping work of adding player starts and a spawn area volume, but I'll leave that to you to do on your own. Once this is completed, rebuild and save your level.

After that you can try out the level, playing as the Germans. Watch where the Soviets spawn during the battle. While you're fighting for the first two objectives the Soviets will spawn at the rear statue. Once you've captured those objectives you should see the Soviets fall back to nearer the rear wall.

I'm sure you can see how this would be a valuable tool when configuring more complicated objective layouts. That pretty much wraps up discussion of the ROSpawnArea actor itself. I do have one last trick up my sleeve that I'll show you in the next section, however, so please read on.

2.5 Temporary Spawn Areas

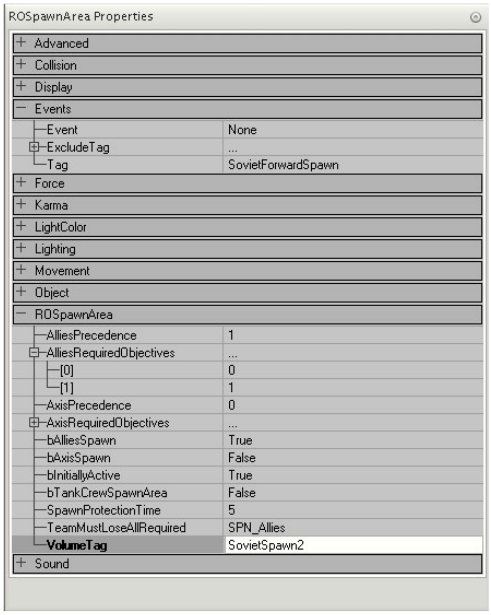
I'm going to start this section a little differently than normal by jumping right into a map. Start up UnrealEd, load the map RO-AdvancedSpawnAreasPt3, and take a look around. You'll see our basic two-objective map-in-a-cube.

Let's look at it from a game play perspective for a moment. Again, the Soviets are defending and the Germans attacking. Almost immediately the more defensive minded among you are probably wondering how the Soviets are going to hold that first objective. The answer is: they probably aren't. There's no way they're going to be able to get from their first spawn to that objective before the Germans.

We already know you could create a forward spawn area, but what it circumstances don't allow you to do that. Or what if you really only want a limited engagement at the first objective? Well, we could certainly make the battle for that first objective tougher if we started the players right there, couldn't we?

How can we do that? With something I call a temporary spawn area, or a "one-time" spawn. This is something I came up with in one of my early Red Orchestra maps. As the name implies, a temporary spawn area is one that is used for only a short period of time. Unlike regular spawn areas, it is not tied to objectives but to some other, external trigger.
A temporary spawn area allows a level designer to place players in the most advantageous position at the start of a level. This eliminates a lot of running (which the players will thank you for) and helps ensure the attackers don't overrun the first objective before the defenders are ready.

Let's start the Soviets at the second statue area for 30 seconds, and then drop them back to the original spawn. We'll do this using a timed trigger, which we'll get to later. First, let's set up the new temporary spawn area.



Add a new ROSpawnArea actor at the second statue and set the following properties:

- bAlliesSpawn: True
- bInitiallyActive: True
- SpawnProtectionTime: 5
- VolumeTag: SovietSpawn2

And, of course, set the Tag in the Event properties group to "SovietForwardSpawn".

Once this is done, go ahead and set up the player starts and the spawn protection volume.

Now, we have to go and tinker a little with the initial Soviet spawn. Open up it's property dialog and set the following properties:

- AlliesPrecedence: 1
- bInitiallyActive: False

This point is a little subtle so I'll explain in detail.

Red Orchestra: Ostfront '41-45 has an explicitly set method for determining which spawn area to use when more than one is available. First of all, available spawns are those that are:

- Active
- Have all their required objective criteria met.

If there are multiple available spawns, then the game uses precedence to determine which to use. The spawn with the highest precedence wins and is used to spawn players.

So, let's say we have two spawns: TemporarySpawn and RearSpawn. RearSpawn starts the game off inactive, so TemporarySpawn is the only one available and is therefore used to spawn players. After some time period, we make RearSpawn active. Now we have two spawns that are available. How do we get the game to use RearSpawn instead of TemporarySpawn? By setting the precedence of RearSpawn to be higher than TemporarySpawn. That's what we've just done. It's really a little sleight of hand with spawn areas.

Now that the spawns are configured, we need some way to make that rear spawn active again. We'll do that using a trigger. We want to use a trigger that will fire after a set amount of time. We'll set the trigger so that it fires after, say, 60 seconds of play. That should be enough time to get the first wave of players to spawn at the forward spawn.

There is actually no specific "timer" trigger actor. What I'm about to show you is one of the most power level designer tools in the UnrealEd toolbox: the ScriptedTrigger.

Open the actor browser and look for Keypoint. Drill down that tree until you find the ScriptedTrigger actor. Place it in the middle of the map (the actual location doesn't matter, it's not position dependent) by right clicking and selecting "Add ScriptedTrigger Here". You'll notice the scripted trigger looks like a switch with an exclamation mark on it.

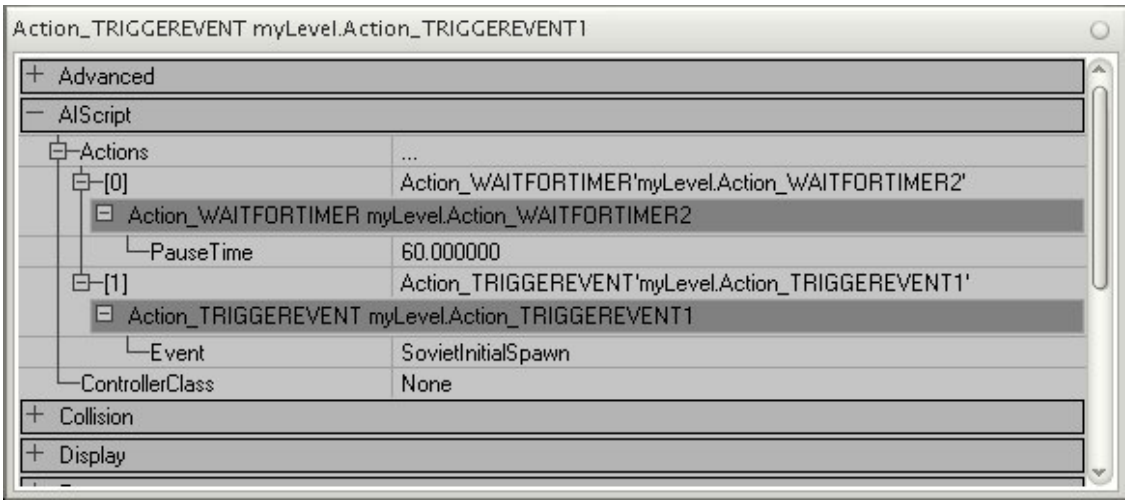
Double click on the scripted trigger to open its property dialog. We're only interested in the AIScript group, so open that next.

I'm not going to go too deeply into the details, as that's fodder for another tutorial. Basically, an AI script is a series of actions that affect the map during play. You have some rough control over the flow of the script and can create simple loops, etc.

We're going to add two actions to this script. The first is going to start a timer for 60 seconds. The second will fire an event to activate the rear spawn. That event will fire after the timer has run out.

Click on the "Add" button twice to add two actions. For the first one, select Action_WAITFORTIMER from the list and press "New". In the PauseTime property, enter 60.

For the second action, select Action_TRIGGEREVENT. The event that we want to trigger is "SovietInitialSpawn". This is the same value we entered as the tag of the rear spawn. When this event is fired, the rear spawn will become active.



A caveat about this little trick: it only works because scripted triggers start processing at the start of every level. That's fine for our purposes but I just thought I'd warn you before you go using this trick in other places.

You can now rebuild and save your level. When you test it, you should see the Soviets spawn at the second statue. Then, after about a minute, you should see them spawn at their end of the cube. Go try it out now.

2.6 Conclusion

It's going to take some time to digest all the information in this tutorial. I've shown you how to set up a simple spawn, a spawn that advances with changing objectives, and how to set up spawns that depend on multiple objectives. Finally, I showed you a little trick that will help you start your levels off with a bang.

Spawn areas are one of the most important things in a Red Orchestra: Ostfront '41-45 level. Their placement and configuration can have a huge impact on game play. The techniques I've reviewed here in this tutorial should help you get the results you want in your own levels. Hopefully, I've also given you a glimpse of what's possible with shifting spawn areas, triggers, and other tricks. Play around with these ideas and I'm sure you'll come up with some interesting and compelling game play hooks of your own.

Part 3: Objectives Tutorial

3.1 Introduction

Virtually every Red Orchestra: Ostfront '41-45 level has more than one objective. These objectives are usually linked in complicated ways so that some are not "enabled" or eligible to be captured until others are first taken. You really won't be able to create the kind of level you want until you can implement this kind of objective network yourself, and that's where this tutorial comes in.

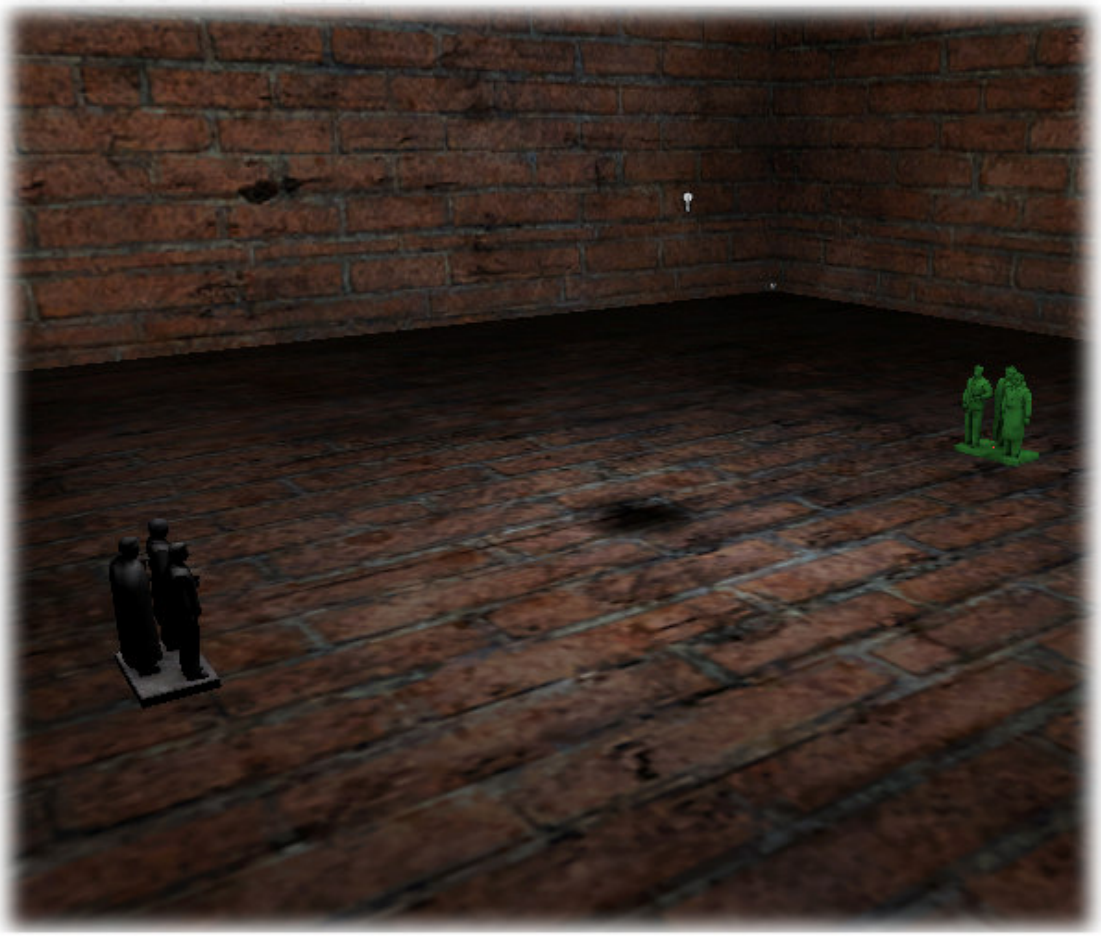
In this tutorial I'll show you how to create a hierarchy of objectives and configure them so you have complete control over their state at any time. We'll be taking a closer look at a couple of the Ostfront map actors, specifically the ROObjTerritory and the MasterObjectiveManager.

For this tutorial, I'll assume you are somewhat familiar with the Red Orchestra level editor and basic Ostfront mapping topics. If not, I'd recommend reading the Quick Start Tutorial first. In fact, I'll be using the map created in that tutorial as a starting point. That map is packaged with the tutorial so you may want to download it even if you are a veteran level designer.

3.2 Simple, Sequential Objectives

Probably the most simple, useful case we can consider is one where there are two objectives with the second depending on the first. That is, we don't want the attacker to be able to capture the second objective until the first has been taken. This sort of arrangement of objectives is quite common in Red Orchestra: Ostfront '41-45 levels, so it's worth taking a look at.

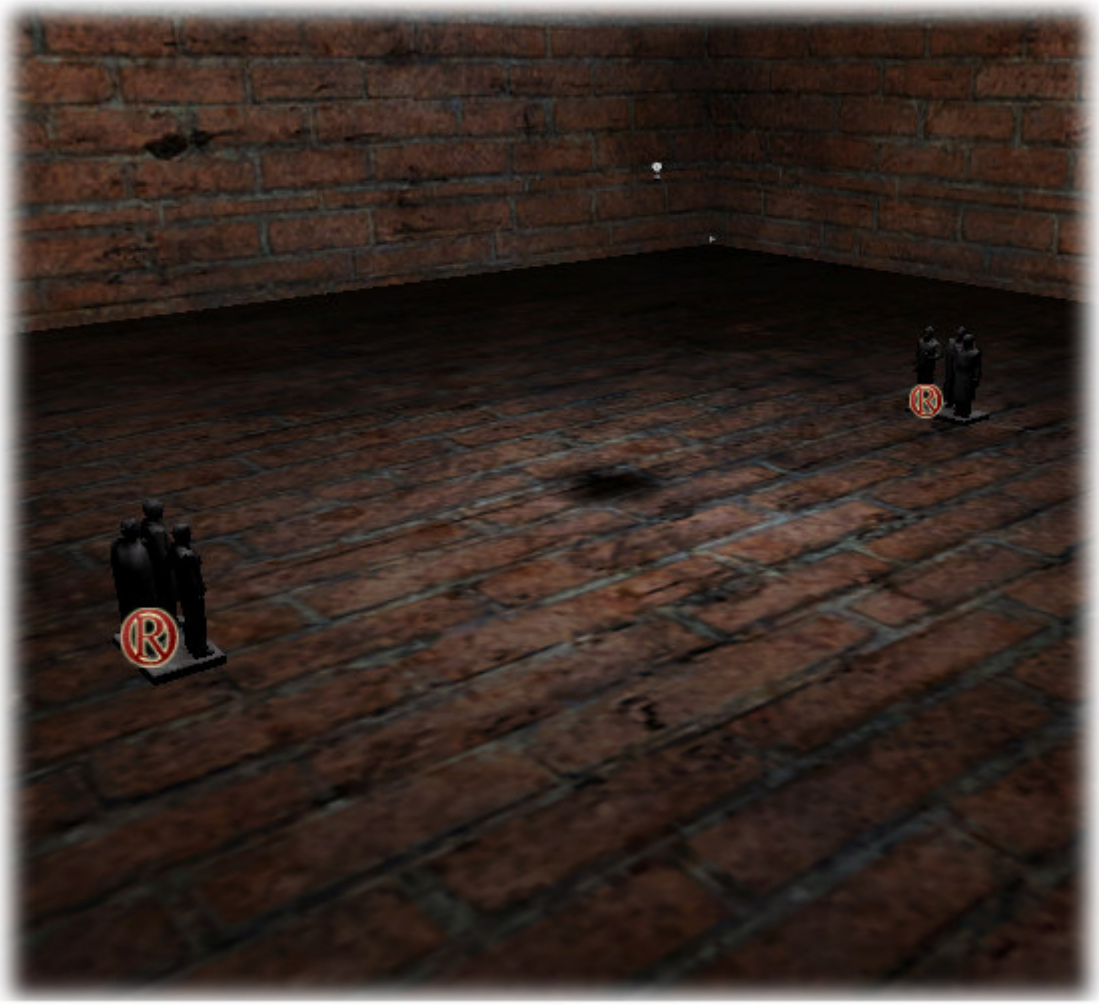
Open the Red Orchestra editor and load the quick start map. If you followed that tutorial, you'll remember we've already placed one objective in this map: a statue in the center of the level. Let's clear all that out so we can start fresh. Select the ROObjTerritory actor and the objective volume and delete them. Select the statue and duplicate it (you can do this by right clicking to bring up the context sensitive menu then selecting "duplicate"). Move one statue until it is about a third of the way across the map from the German spawn by selecting it and dragging it in the top view while holding down the control key. Move the second until it is a similar distance from the Soviet spawn. Once you're done, you should see something like this:



Now, open the actor browser and find the ROObjTerritory actor. This can be found under:

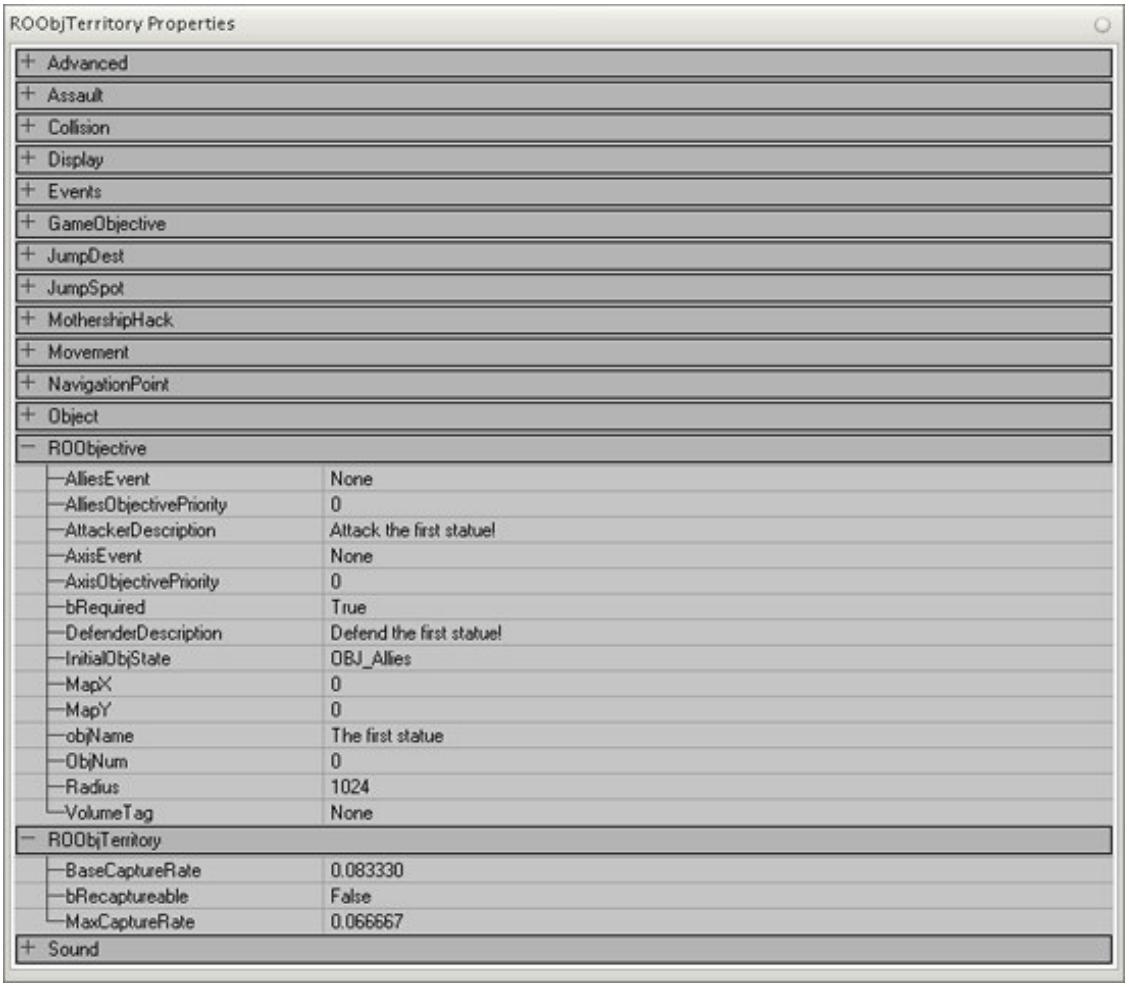
NavigationPoint->JumpDest->JumpSpot->GameObjective->ROObjective

Select ROObjTerritory and place two into the map, one at each statue. Once you're done, your level should look like this:



The next step is to configure the objectives. First, let's talk a bit about the order of the objectives. Since the Germans are attacking in this map, we'll want to make the objective that is closest to their spawn the "first" one and the one that's closest to the Soviet spawn the second. The first spawn will be "enabled", or eligible for capture, when the level starts. The second objective will not be enabled until the first is captured.

Double click on the first objective (the one closest to the Germans) to open its property dialog then open the ROObjective and ROObjTerritory groups. You should see something that looks like this:



Set the following properties:

- AttackerDescription: Attack the first statue!
- bRequired: True (already set).
- DefenderDescription: Defend the first statue!
- InitialObjState: OBJ_Allies
- objName: The first statue
- ObjNum: 0 (already set)
- Radius: 512

And in the ROObjTerritory group, set bRecapturable to False.

So, what did we just do? Well, we did some housekeeping on the objective, but you may have missed the most important properties. By setting bRequired to True we told the game that this objective must be captured in order for the Germans to win the level. We also told the game the objective starts out in Soviet hands. Setting the ObjNum property to 0 indicates that this is the first objective in the level (we start

counting from zero). Our level isn't very big, so we'll trim down the area that each objective covers to a radius of 512 UnrealEd units.

Finally, setting bRecapturable to False lets the game know that the objective cannot be recaptured once the Germans have taken it.

Let's go ahead and configure the second objective as well. Set the following properties:

- AttackerDescription: Attack the second statue!
- bRequired: True
- DefenderDescription: Defend the second statue!
- InitialObjState: OBJ_Allies
- objName: The second statue
- ObjNum: 1
- Radius: 512
- bRecapturable: False

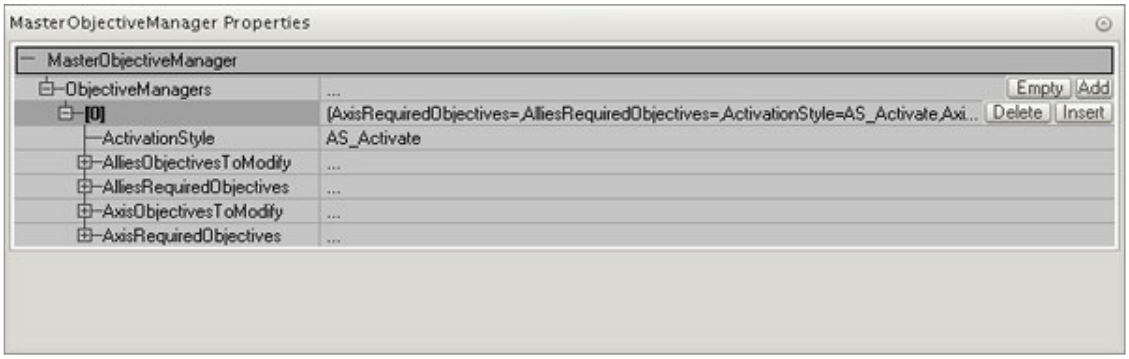
But wait, we need to do one more thing. Open up the GameObjective property group and look for the property bInitiallyActive. Set this property to False. This tells the game that this objective should not be active (or eligible for capture) at the start of the level.

Now rebuild and save the level (say as RO-AdvancedObjectives).

If you were to play the level at this point, you'd see that both the objectives are initially enabled (eligible for capture) at the start of play. That's not what we want. We want only the first objective to be enabled and the second to come into play when the first is captured.

We can't do this without additional help. That help comes in the form of the MasterObjectiveManager. Find it in the actor browser and place on right in the middle of the map. Once that's done, double click it to bring up the properties dialog. Not very impressive at the moment, is it? Basically, the MasterObjectiveManager is empty and there's nothing to see.

Click on the ObjectiveManagers property and then click on the "Add" button. Now you should see something like this:



The MasterObjectiveManager essentially acts as a container for instructions on how to control the state of objectives. Each instruction is designed to change the state of one or more objectives under certain conditions. I won't lie to you, sometimes it really is as complicated as it sounds, but lets go through our simple case here and see how it works.

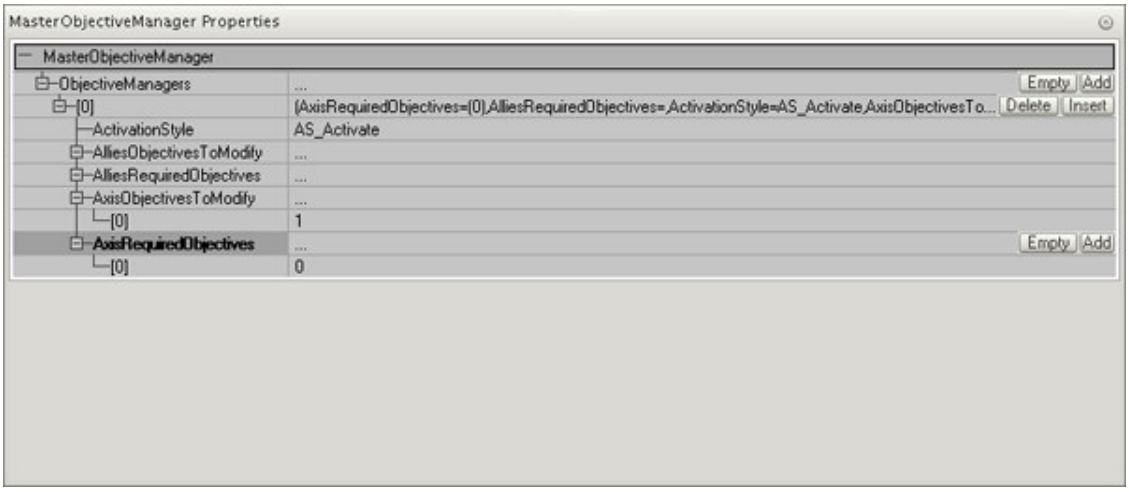
We need to introduce one state change for our objectives: when the Germans capture the first objective, the second is enabled. Since our objectives aren't recapturable, we don't have to worry about the reverse case.

Let's go ahead and set up this state change. Working with the ObjectiveManager we just created, set the following properties.

- ActivationStyle: AS_Activate
- AxisObjectivesToModify: 1
- AxisRequiredObjectives: 0

This tells the objective manager to activate, or enable, the objective with ObjNum = 1 when the Germans hold the objective with ObjNum = 0.

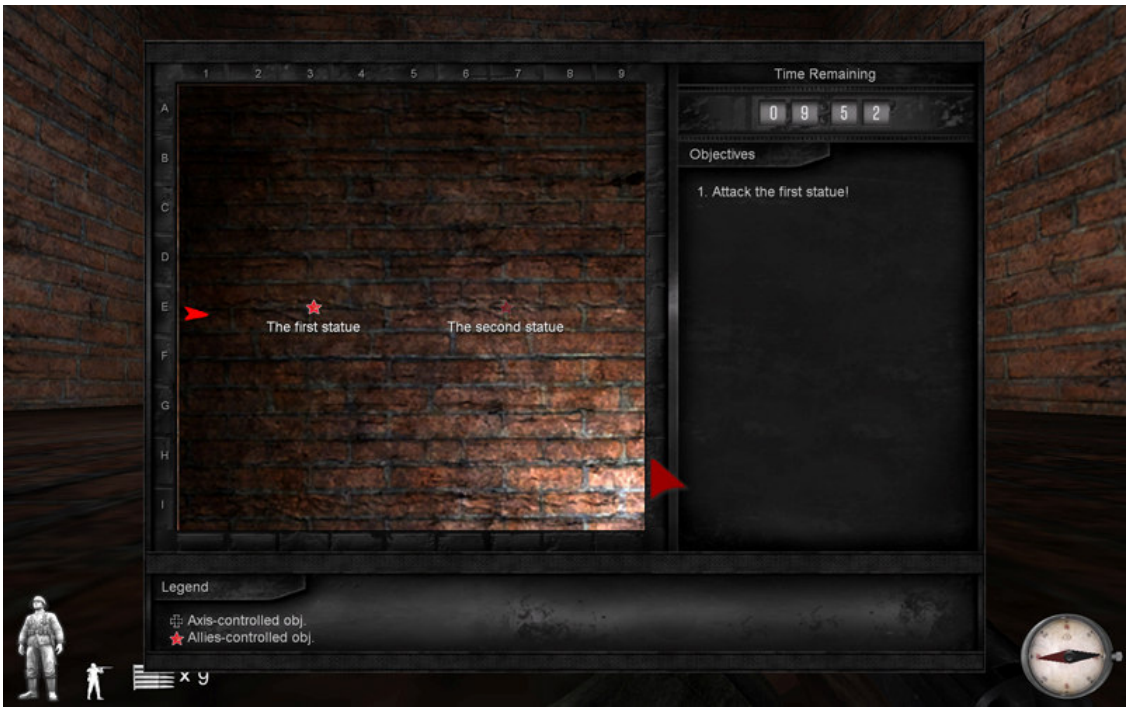
If you've done everything correctly, your MasterObjectiveManager should look something like this:



If you made a mistake it's very easy to go back and try again. Simply click the "Delete" button on the ObjectiveManager entry to remove it, add a replacement and start over.

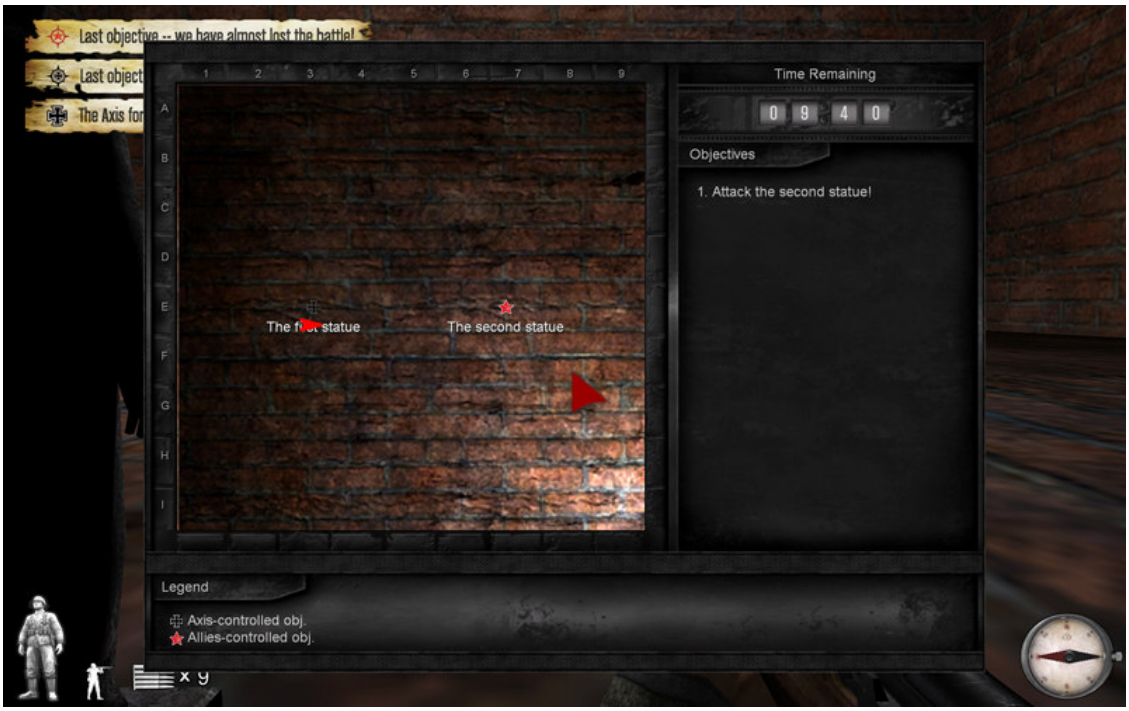
Rebuild and save the map. This is a good time to test what we've done, so take a break and play test the level.

During play and at the start of the level, if you look at the objectives screen, you should see something like this:



You may notice two things. The first is that the red star that indicates the second objective is shaded. You may also notice that the text for the second objective doesn't appear in the objective list on the right. This is because the second objective is not activated at the moment.

Now, enter the game as a German and capture the first objective. Your objective screen should now look something like:



Notice that the second objective is now active. Notice also the changes in the objective list on the right side of the screen.

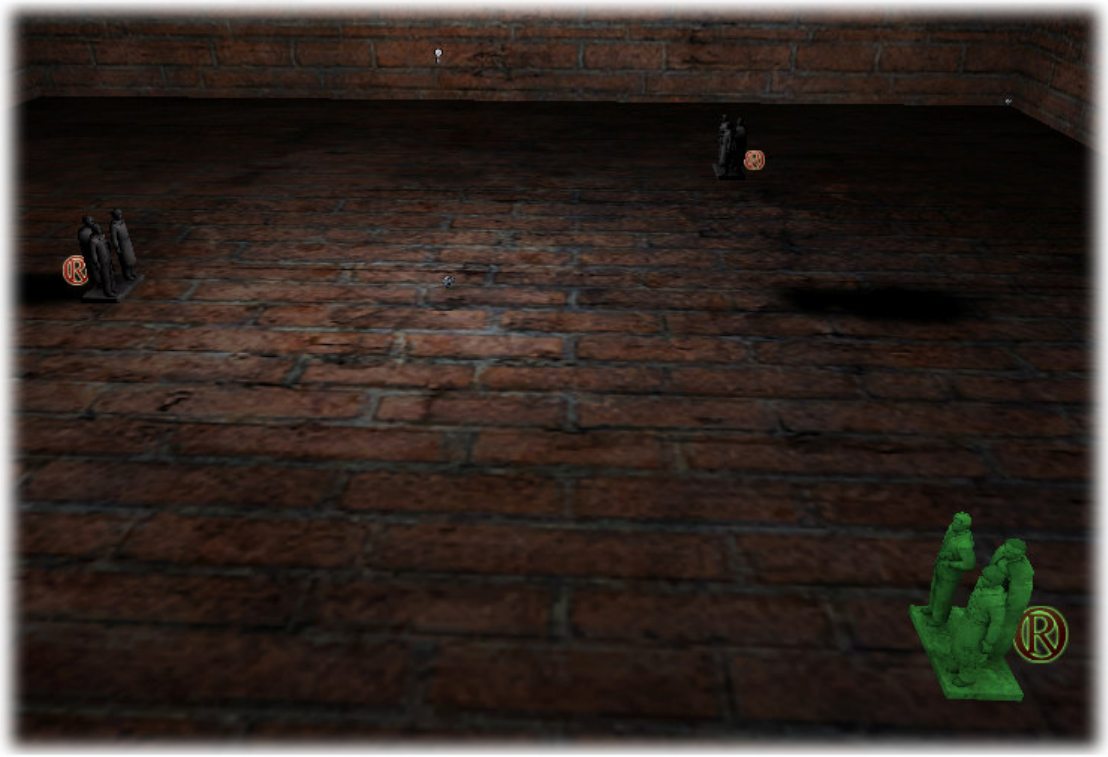
Congratulations, you have just configured two objectives so they are sequentially linked. You have just added a powerful new tool into your mapping arsenal that will allow you to control game play and flow in your future levels.

Let's continue and try something a little more complicated, just to cement the concepts.

3.3 Complex Objective Arrangements

In this section we'll explore a slightly more complicated arrangement of objectives. We'll use three objectives and arrange them in a triangle, with two forward and one back. The two forward objectives must be captured before the rear one is activated. Finally, we'll set all the objectives so they can be recaptured.

Start up UnrealEd again and open our test map. We're going to need another objective, so just duplicate the one nearest the German spawn (including the ROObjTerritory actor). After that, move the statues and actors so they're arranged in a triangle. The two forward objectives should be towards the German spawn. When you're done, it should look something like this:



Now, set the following property values for each objective.

For objective 1 (either of the objectives nearest the German spawn):

- AttackerDescription: Attack the first statue!
- bRequired: True
- DefenderDescription: Defend the first statue!
- InitialObjState: OBJ_Allies
- objName: The first statue
- ObjNum: 0
- Radius: 512

And in the ROObjTerritory properties, set bRecapturable to True.

For the second objective (the other objective nearest the German spawn):

- AttackerDescription: Attack the second statue!
- bRequired: True
- DefenderDescription: Defend the second statue!
- InitialObjState: OBJ_Allies
- objName: The second statue
- ObjNum: 1
- Radius: 512

And again, set bRecapturable to True.

Finally, for the last objective (nearest to the Soviet spawn):

- AttackerDescription: Attack the third statue!
- bRequired: True
- DefenderDescription: Defend the third statue!
- InitialObjState: OBJ_Allies
- objName: The third statue
- ObjNum: 2
- Radius: 512

Now, we could set this objective so that it can be recaptured, but there's really no point. If the Germans ever take this objective the game is over anyway, so we can leave it with the default setting. On the other hand, we want this objective to be deactivated at the start of the game. Open the GameObjectives properties and set bInitiallyActive to False.

Rebuild and save the map.

We can now move on to mapping out the state changes for the objectives. What state changes do we have to worry about?

1. When the Germans capture objectives 1 and 2, we need to active objective 3.
2. When the Soviets capture either objectives 1 or 2, we need to deactivate objective 3.

Notice that we need that second state change because the front objectives can now be recaptured. It wouldn't do to activate the rear objective and then forget to deactivate it if the Soviets put up a counterattack.

So, let's set this up. Double click on the existing MasterObjectiveManager to access it's property dialog. Delete the existing ObjectiveManager entries so we can start fresh. Press the "Add" button to create a new ObjectiveManager. We'll map the first state change here. Set the following properties:

- ActivationStyle: AS_Activate
- AxisObjectivesToModify: 2
- AxisRequiredObjectives: 0, 1

What we've done here is tell the game that when the Germans capture the objectives with ObjNum set to 0 and 1, then activate the objective whose ObjNum is 2.

Create another ObjectiveManager and set the following properties:

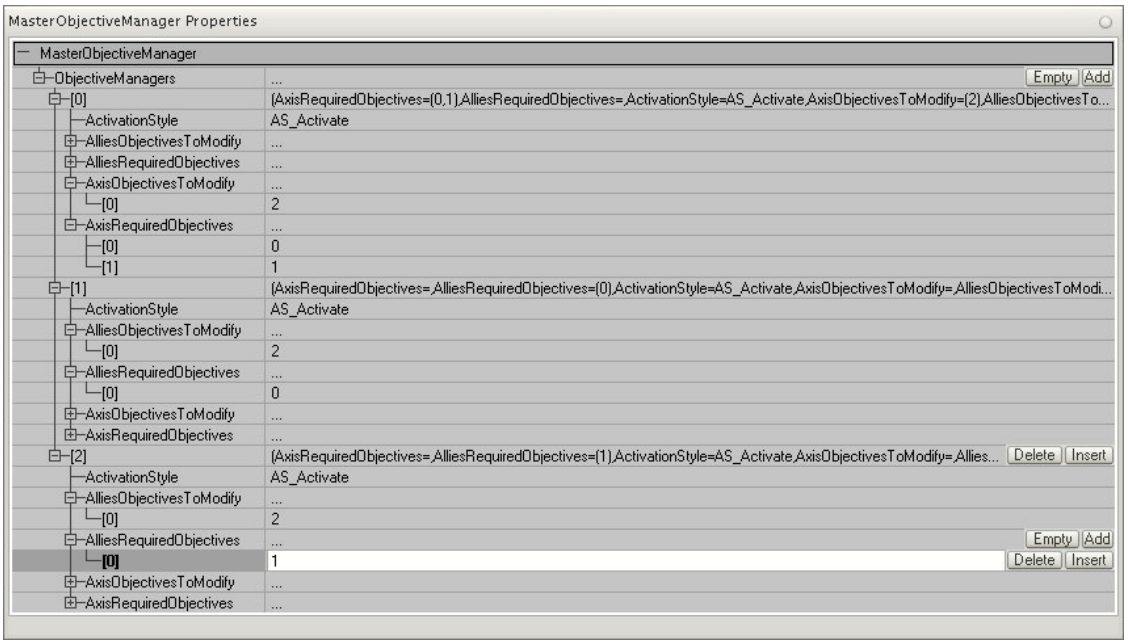
- ActivationStyle: AS_Deactivate
- AlliedObjectivesToModify: 2
- AlliedObjectives: 0

And create one last ObjectiveManager and set it as follows:

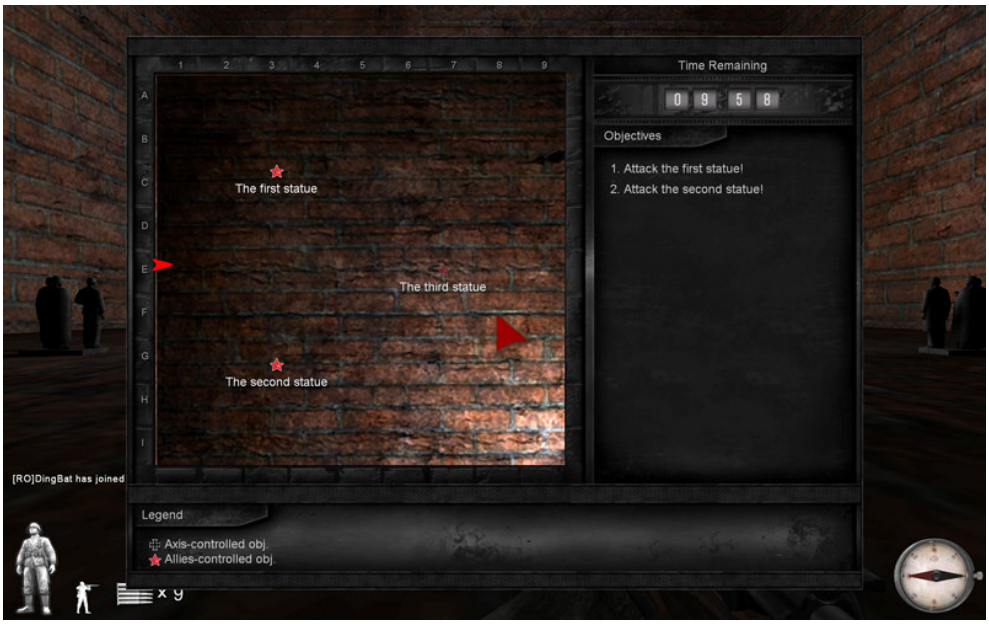
- ActivationStyle: AS_Deactivate
- AlliedObjectivesToModify: 2
- AlliedObjectives: 1

Why did we need two ObjectiveManagers to map that state change? Because the last objective must be deactivated if the Soviets capture the first OR second objective. The OR really means there are two state changes hidden in one.

Once you're finished, the MasterObjectiveManager should look like this:

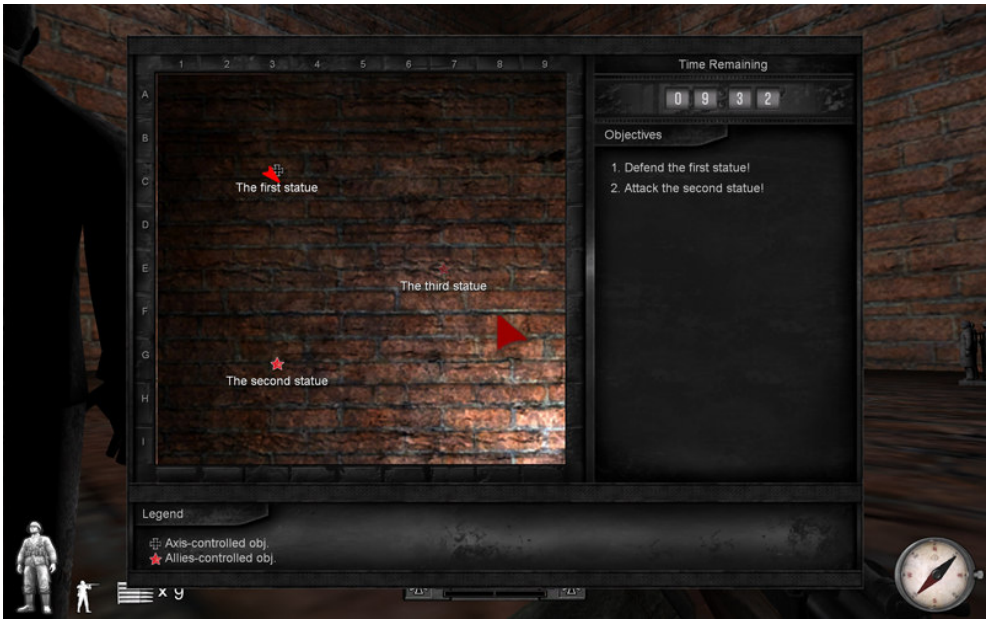


Now go start the game and try out the level. When you're able, take a look at the objectives screen. It should look something like this:



Notice that the third statue is not active and there are only two objectives listed on the right. You can't capture the third objective until the first two are taken!

If you go ahead and capture the first statue, you'll see that the objective screen has changed:



Notice that the third statue still isn't active. That's what we want. You may also notice that the objectives now tell you that you must defend the statue that you just captured. Watch for sneaky Soviets trying to reclaim lost ground.

Go ahead and capture the second statue and take a look at the objectives again:

Finally, you can see that the third statue is now active. You also have two statues to defend as well, so that third statue may not be so easy to take.

As a little experiment, you could switch to the Soviet side and recapture one of the statues. You should see the last statue become inactive again.

3.4 Conclusion

In this tutorial, I've shown you how to set up two very common objective arrangements. The majority of situations you will face when designing your levels will most likely involve some combination of these types. Some objective arrangements can be fiendishly complicated, but if you take it slow and thing about the state changes that need to occur, you won't go wrong. Just remember the deactivation state changes when objectives are recapturable.

That's it for this tutorial. I will include the two maps we've worked on here so you can refer to them later or use them in future experiments. I hope you enjoyed this look at advanced objectives in Red Orchestra: Ostfront '41-45 and that it will help you create amazing levels in the future. Good luck.

Part 4: Overhead Map Tutorial

4.1 Introduction

A battlefield is a pretty confusing place for most soldiers, especially given the fact that at least one side is usually in territory not familiar to them. Red Orchestra: Ostfront '41-45 provides an overhead map for players to help them orient themselves in the virtual battlefield and find the fight.

In this tutorial, I'll show you how to add an overhead map to your level and how to tune it so that it accurately reflects player movement. At this point, I'm assuming you have at least a basic knowledge of the Red Orchestra level editor. If not, I'd suggest taking a look at the Quick Start tutorial before returning to this text.

As with all my tutorials, I'm providing the test level that will be used throughout this document. It serves as a decent, common starting point that lets us get right down to working with the overhead map components. I've also provided a sample overhead map shot in case you don't want to be bothered doing it yourself.

So, let's get started.

4.2 The Overhead Map Components

Obviously, the most important part of the overhead map is the map picture itself. In this section, I'll show you how to make this. It will require a bit of work in the game itself, plus some sort of graphic manipulation tool like Photoshop. If you don't want to be bothered with all this, then just use the texture I've provided and move on to the part two of the tutorial.

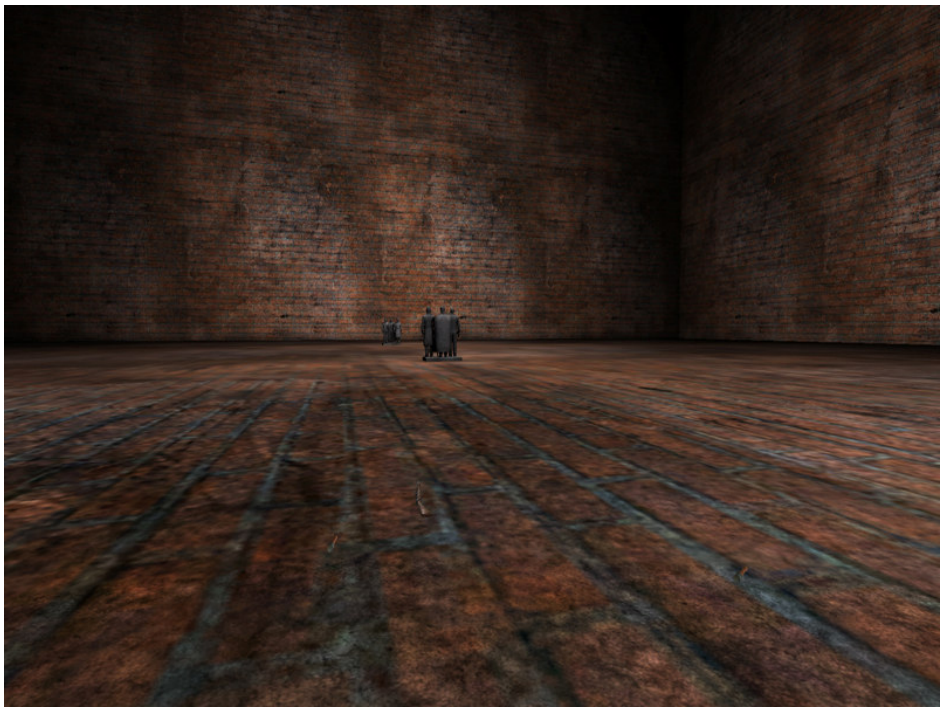
As I said, we're going to want to actually play this level so the first step is to make sure our tutorial map is in the playable maps areas of the Red Orchestra: Ostfront '41-45 installation area. Find the file "RO-OverheadMap.rom" that was included with this tutorial and copy it to the maps folder in the Red Orchestra installation.

The overhead map is actually a pretty simple concept: we're going to take a screen shot of the level looking down on the playing field. The concept is simple but, unfortunately, the execution is a little more complicated.

First, start up the game and enter the level. Remember to turn bots off before playing. Once the game has started you should see something like this:

We can't take a screen shot in our current state because the view is cluttered with our HUD and weapons. Let's take care of that.

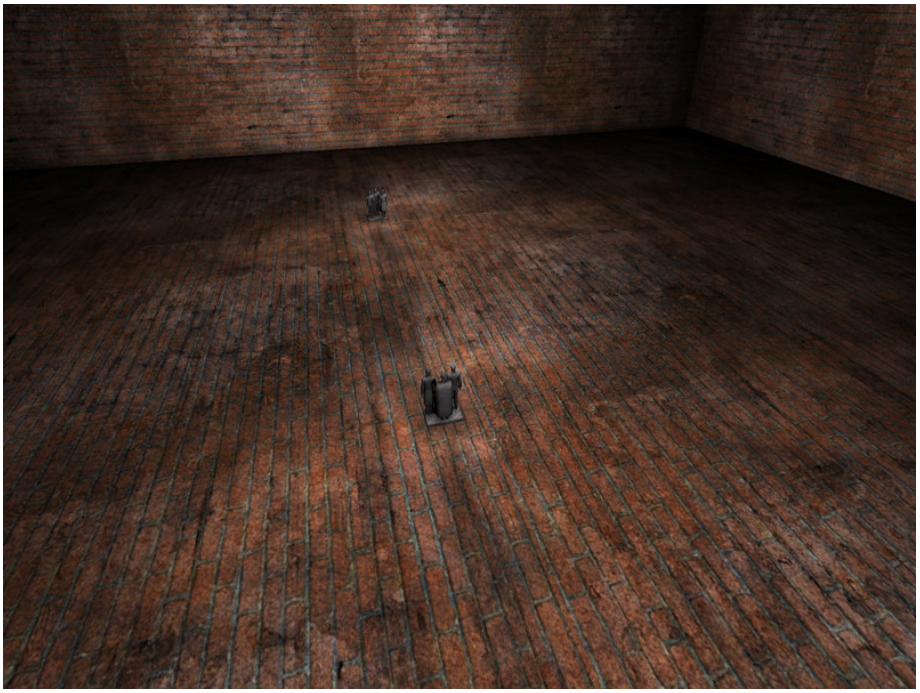
First, drop all your weapons (you should have a key bound to this in the game). Once you've done that, open the console by hitting the tilde key (~). In the console, type "showhud 0". This will turn your HUD off. You should now see something that looks like this:



Much better.

Now, we can't take a top down screenshot at the moment, can we? We're stuck on the ground. Let's fix that.

Again in the console, type "fly". The game will respond by telling you that you feel lighter. You'll notice, if you move around now, that you're no longer stuck on the ground. You'll also notice your movement is agonizingly slow. Luckily, we can fix that too. Open the console one last time and type "slomo 10". You should now be able to move anywhere about the level at a decent speed.

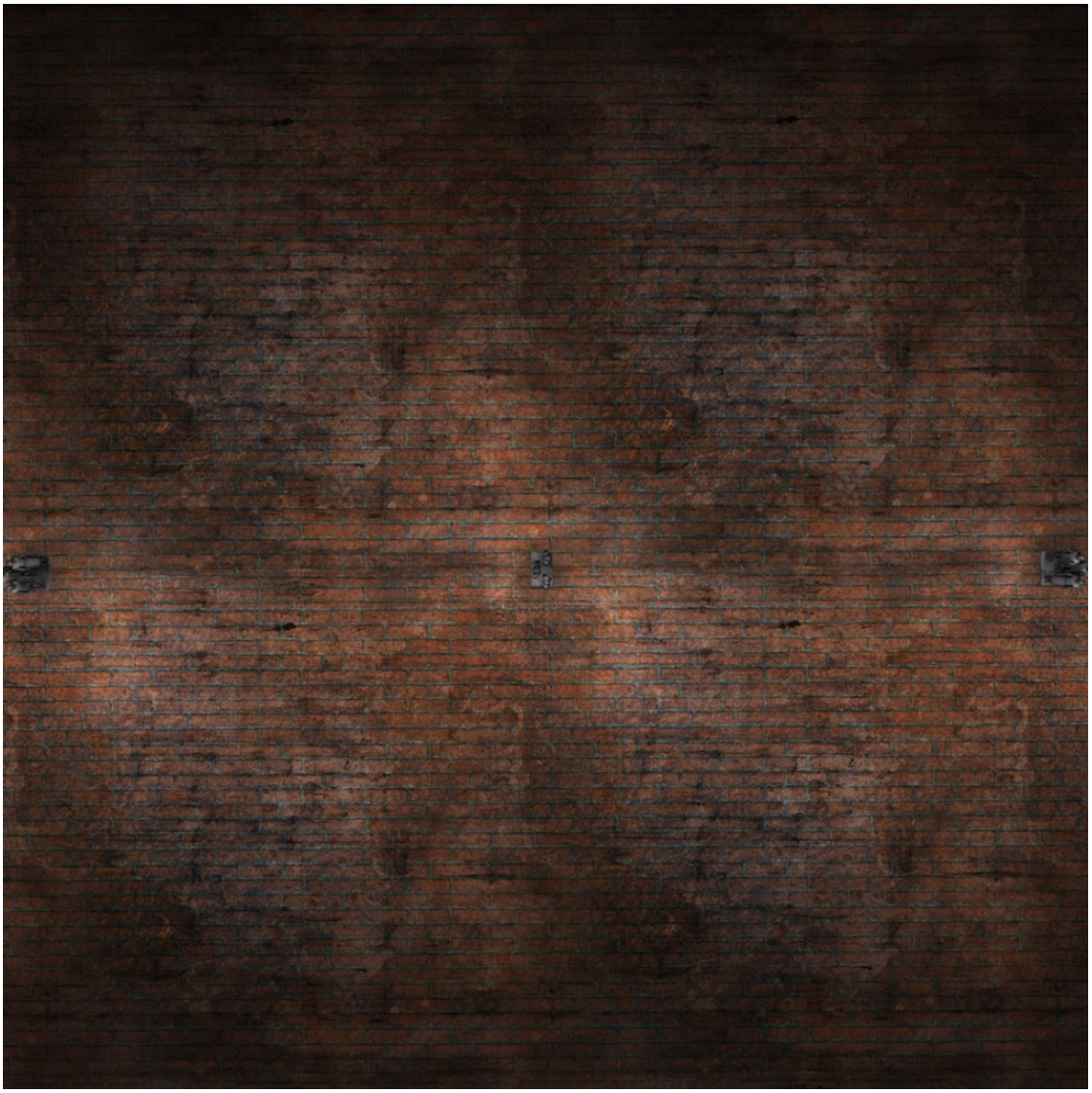


There's one last thing to do before we take our screen shot. Right now, your screen resolution is likely 1024x768 or 1280x1024 or something similar. This is no good for our purposes as we're going to need a texture 512 pixels on a side.

Force the game into windowed mode by simultaneously pressing the control, alt, and enter keys. Then, in the console, type "setres 512x512". You should see the game reconfigure itself into a square. Now we're ready to take a screen shot.

Position yourself above the playing field at a point directly above the middle statue and just high enough that the side statues are in view. Hit f9 to take the screen shot. That's it, we're done.

Here's what our overhead map shot should look like:



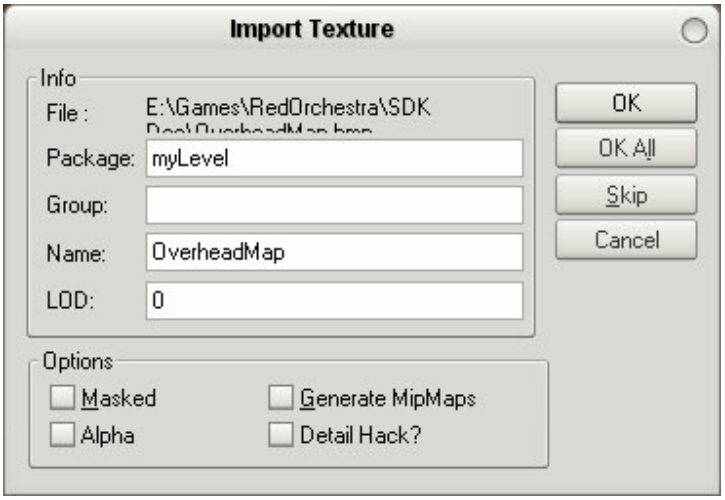
You can restore your game to its original configuration by doing the following:

- Maximize the game screen.
- Type "slomo 1" in the console.
- Type "fly" in the console (this is especially fun if you're still quite high in the air).
- Type "showhud 0" in the console.

4.3 Adding the Overhead Map

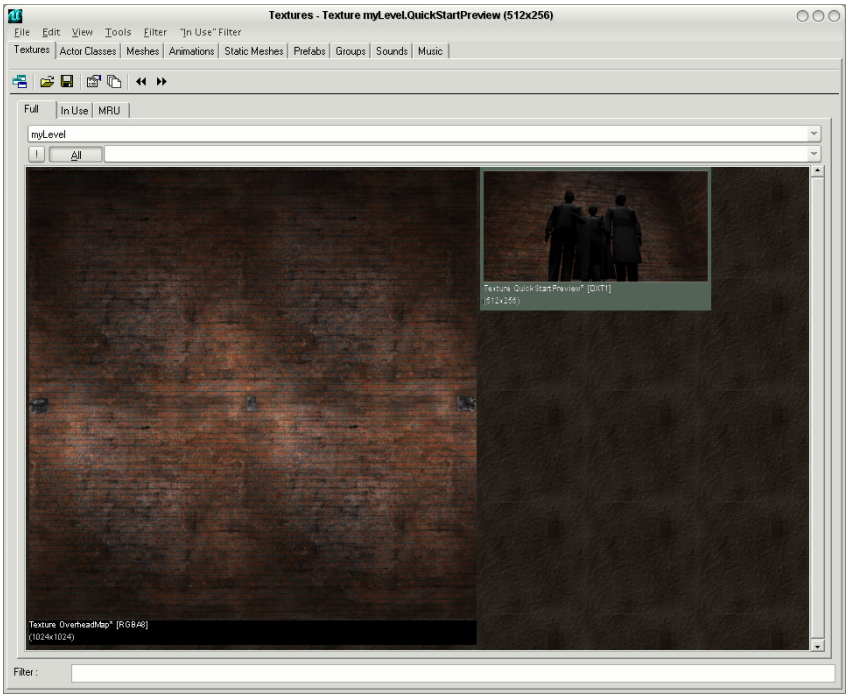
Now that we've got our overhead map shot, we need to get it into the level editor somehow.

Fire up the editor and open the map "RO-OverheadMap". Open the texture browser and select "Import" from the File menu. Locate the file that contains your overhead screen shot and click Open. You should now see a dialog that looks like this:



A quick explanation is in order. The Red Orchestra level editor organizes textures into collections called "packages". These packages are usually saved in separate files that are found in the textures directory of Red Orchestra: Ostfront '41-45. The exception to this rule is the "myLevel" package. This package is "private" and is bundled with the level itself. You should use the myLevel package for textures and art that are specific to your level, and the overhead map certainly qualifies.

You can call the overhead map anything you want (in this case, I've called it "OverheadMap"). Make sure the "Generate MipMaps" checkbox is cleared before you proceed. Once this is done, you can hit "OK" to import the map.



You should notice that the overhead map now shows up in the texture browser.

If you look closely at the text at the bottom of the texture you will see it's name (OverheadMap) and it's size (512x512). You'll also see that it is currently in a format called "RGBA8". I don't want to get into this too much, but suffice it to say that this is a wasteful format for our map. We have to keep an eye on imported textures, especially in the myLevel package, since we can bloat the size of our map file quite considerably if we're not careful.

We can afford to compress the overhead map so right click on the texture in the browser. Select "Compress" from the list and choose "DXT1" compression. When the compression is complete you should notice the new format in the texture details.

You can now rebuild and save your map. Once that's done you can go ahead and play it as well. Once in game you will quickly notice that the map isn't quite correct. This is because the screenshot we took isn't the same size as the map itself. We'll correct that in the next part of this tutorial.

4.4 The Map Boundary Actors

If you tried to play our test map and check out the overhead map, you undoubtedly noticed that it's messed up. The objective isn't in the right place and the map doesn't accurately show the players position.

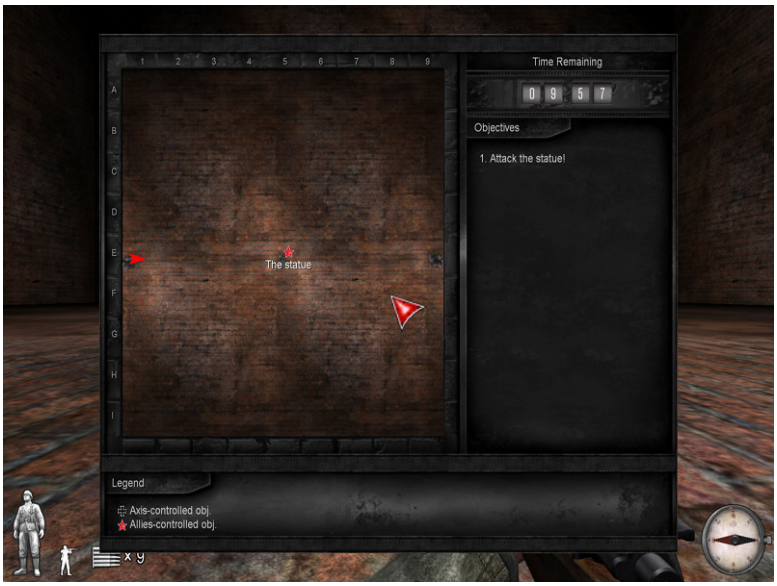
The reason for this is that we haven't set out the boundaries of the map in the level itself. Our screenshot only includes a portion of the map. We have to tell the game where the boundaries of our overhead map are in the actual level. This is done with a pair of map actors: ROMapBoundsNE and ROMapBoundsSW. As you might suspect, these actors mark the northeast and southwest corners of the overhead map. Once we place these actors in the map our overhead map should work just fine.

Start up the Red Orchestra level editor again and reload our test map. Open the actor browser and fine the ROMapBounds actor group.

Now, before we go any further, you'll have to decide where north is on your map. When I took the screen shot that is used in this tutorial, I decided that north was "up" in the editor. That puts it to the left of the Germans when they spawn, and to the right of the Soviets.

With that decided, you should be able to find the northeast and southwest corners. Try to find points that approximate the edges of your overhead map screenshot. Place the ROMapBoundsNE actor in the northeast corner but selecting it in the actor browser, then right clicking on the appropriate place in the map and selecting "Add ROMapBoundsNE Here". Do the same thing for the ROMapBoundsSW actor in the opposite corner.

Once that's done you can rebuild and save your level. Now, if you play the level, your overhead map should display correctly. In fact, it will look something like this:



4.5 Conclusion

The overhead map is an important part of any Red Orchestra: Ostfront '41-45 level. In the course of a round of play it is referred to by players to orient themselves, to see where artillery strikes have been targeted and to coordinate attacks. In this tutorial, I've shown you how to create a simple map using a screenshot, how to add the screenshot to a level, and how to properly configure the map using the map boundary actors.

Really, the mechanics of establishing an overhead map are pretty straightforward. The real challenge is to create a map that is attractive as well as functional. The screen shot technique I've shown you will work for most of your levels, but you don't have to stop there. The early Red Orchestra mod maps were made to look like hastily hand drawn maps, for example. The look of your overhead map is up to you.

Part 5: Artillery Tutorial

5.1 Introduction

"Artillery adds dignity to what otherwise would be a vulgar brawl"
Unknown

Sometimes more than just small arms are required to get the job done. For those really tough assignments, it's always good to have a battery of 105mm howitzers at your back. Luckily, Red Orchestra: Ostfront '41-45 provides the level designer with the ability to add artillery to either, or both, sides in a map.

In this tutorial, I'll show you how to configure the artillery assets for each side. I'll show you how to add radios so that field commanders can actually call in artillery strikes. Finally, I'll discuss how to set the map bounds actors so the sounds of the guns is music to your ears while in game. None of this is particularly difficult to set up, but I am assuming at this point that you have at least a basic knowledge of the Red Orchestra level editor and map development in general. If not, I'd suggest taking a look at the Quick Start tutorial before returning to this text.

As with all my tutorials, I'm providing the test level that will be used throughout this document. It serves as a decent, common starting point that lets us get right down to working with the artillery components.

So, let's get started.

5.2 Adding the Guns

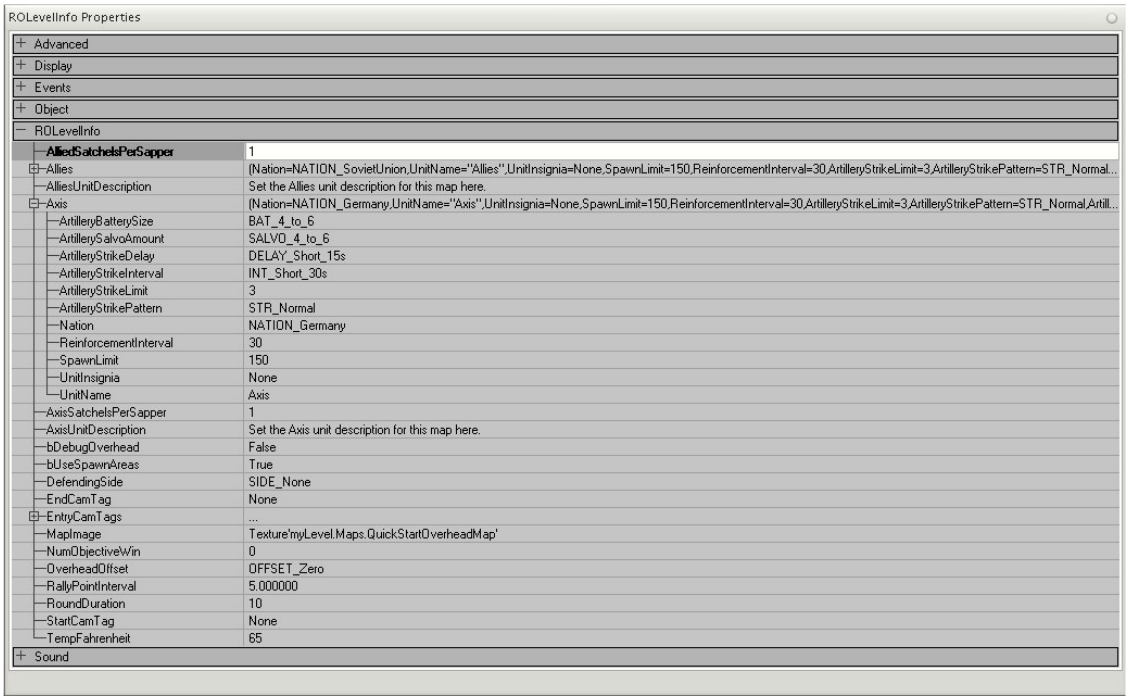
Deciding to add guns to your map is a design decision. It shouldn't be taken lightly. When working on your own levels, you should ask yourself these questions before proceeding with artillery:

1. Is it realistic for this force to have artillery?
2. Is it necessary for this force to have artillery?

If the answer to these questions is yes, then you are probably correct in adding artillery to your level.

But that doesn't matter to us now. For the purposes of this tutorial, lets add artillery to the German side. Start up the Red Orchestra level editor and open the tutorial map "RO-Artillery".

In Red Orchestra maps, the artillery for either side is configured in the ROLevelInfo actor. You'll find that in the extreme top left hand corner of the tutorial map (remember, it looks like a horses head). Double click on the actor to open its property dialog and then click on the Axis property group. You should see something like this:



At the top of the Axis group you'll find several properties related to artillery. I'll explain them first before we continue.

Before we do that, lets introduce the concepts of fire missions and salvos. A fire mission describes a request for fire support that comes from the field. Each fire mission may include more than one salvo. A salvo is when all the guns in the battery

fire. So, for example, if a field commander requests a fire mission of 3 salvos, the guns in the battery will each fire three times.

Property	Description
ArtilleryBatterySize	Sets the number of guns in the battery
ArtillerySalvoAmount	Sets the number of salvos, or the number of times each gun fires, in a fire mission.
ArtilleryStrikeDelay	Sets the delay, in seconds, between the time the fire mission is requested and the time the guns actually fire.
ArtilleryStrikeInterval	Sets the delay, in seconds, between salvos in a fire mission.
ArtilleryStrikeLimit	Sets the maximum number of fire missions that may be requested in a round.
ArtilleryStrikePattern	Sets patterns for the rounds in the fire mission.

Now, for the most part, you're going to want to keep the battery size fairly low. Battles in Red Orchestra: Ostfront '41-45 include less than a full platoon of men and it was pretty rare that a unit this size would be able to call on a complete battery for fire support. You'll also probably want to keep the salvo number low. The delays in the artillery strikes should depend on the type of guns and the side. Delays for large calibre guns were typically longer as it takes quite a while to clear and reload these big guns. Additionally, the communications were typically not as reliable in the Red Army as in the Wehrmacht, so the soviet side should experience longer delays.

For this map, let's say we're simulating a battery of four 105mm guns. It's got plentiful ammo and it's dedicated to our support. Set the properties to the following values:

- ArtilleryBatterySize: BAT_4_to_6
- ArtillerySalvoAmount: SALVO_2_to_3
- ArtilleryStrikeDelay: DELAY_Short_15s
- ArtilleryStrikeInterval: INT_Short_30s
- ArtilleryStrikeLimit: 3
- ArtilleryStrikePattern: STR_Normal

So far, so good. You can rebuild and save the map now. However, before we can try out our new toys, there is still some more work to do.

5.3 Artillery and Leaders

In Red Orchestra: Ostfront '41-45 only leaders have the authority to call in artillery strikes. That means we'll have to add at least one leader role to the German side.

Start up the Red Orchestra editor and re-open your tutorial map. Open the actor browser and find the RORoleInfo group of actors. Drill down into the ROGE_Standard_Heer sub group. Find the role ROGESquadLeaderH and add it to the map by right clicking and selecting "Add ROGESquadLeaderH Here".

That's it. Yes, really. It's that simple. Still, even though we've added a leader, we have no way for him to request artillery. Let's fix that next.

5.4 Radios and ROArtilleryTriggers

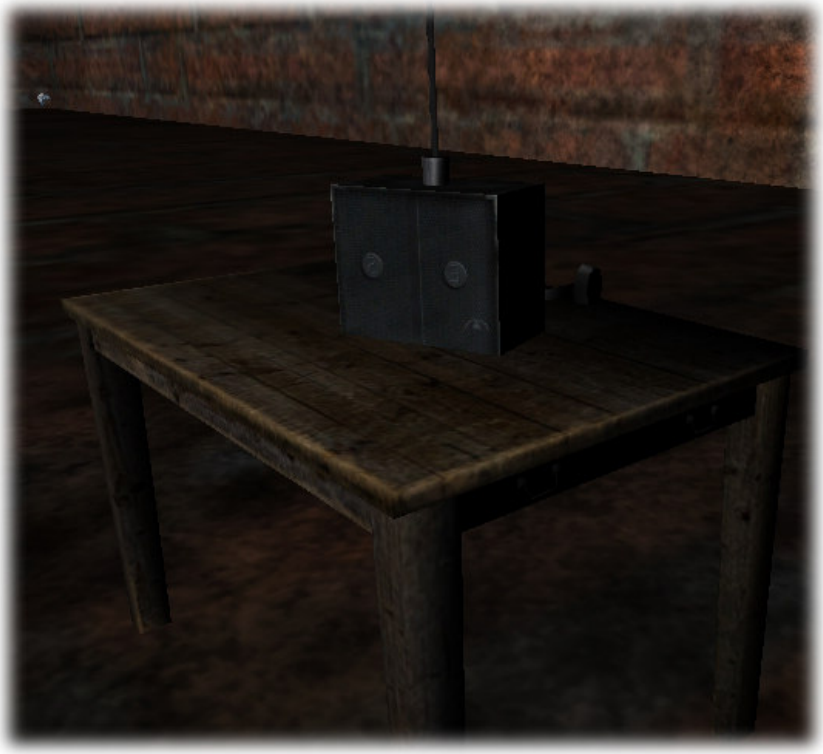
In order to request artillery in game we need two things:

- A radio (static mesh)
- A trigger (specifically, a ROArtilleryTrigger).

First, lets set up the radio. Once again, open your tutorial map in the editor. Launch the static mesh browser and click on the "Open Package" button (which looks like an open file folder).

Find and open the static mesh package called "FurnitureSM". We're not going to just put our radio on the floor. We'll need a table or something first. Search through the meshes until you find "Table2". Place that mesh in the level by right clicking near the German spawn and selecting "Add Static Mesh: "FurnitureSM.Tables.Table2 Here". You may have to play with the positioning of the mesh a bit so that it sits squarely on the floor.

Now open the package "MilitarySM". Find the mesh "radio" and add it to the level, right on top of the table we just added. Play around with the positioning until it's correct. When you're done, you should have something that looks like this:



Now switch to the actor browser and open the Triggers group. Find the actor "ROArtilleryTrigger" and add it to the level in the same place as the radio.

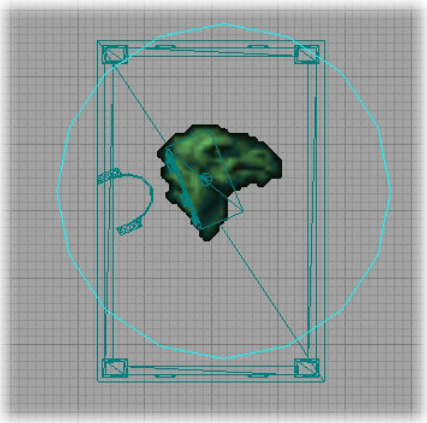
Double click on the new actor to open its property dialog then click on the ROArtilleryTrigger group. You should see something like this:

As you can see, there's not much to configure. The Message property allows you to set a message that players will see when they get within range of the radio. Set that to "Use the radio to request artillery from HQ."

The TeamCanUse property determines which team can use the radio. We don't want any sneaky soviets using this radio against us, so set that to AT_Axis.

Now, speaking of the radio range, we should look at that as well. All triggers fire when a player gets within a certain range. The ROArtilleryTrigger fires when a player presses the "Use" key within its range. The default value for range can be a little small so let's extend it bit.

First, right click on the upper bar in the Top view. Select Actors and then Radii View. We've just enabled the top view so that it shows the collision radius for any selected actor. Click on the ROArtilleryActor. You should see something like this:



That blue circle around the actor shows it's collision radius. It's a little small (someone would have to stand right in front of the table to activate it), so let's extend it.

Open the property dialog again and click on the Collision group. Near the bottom of the group is a property called CollisionRadius. Enter a value of 128. You should see the blue circle grow quite a bit. That's good enough for our purposes.

You can again rebuild and save the level. In fact, we've done enough to test it. Fire up Red Orchestra: Ostfront '41-45 and try out your tutorial level. Remember you must play as a leader in order to use artillery. You'll have to plot an artillery target by using your binoculars and clicking where you want artillery to land. Only then will the radio allow you to request a fire mission.

There's only one last little thing to take care of and that's synchronizing the artillery with the map boundary actors. We'll look at that in the next section.

5.5 Artillery and Map Boundary Actors

You may (or not) have noticed if you played our tutorial level that the sound wasn't quite in synch with the visuals. This has to do with the placement of the map boundary actors in the level. In order for the artillery to operate properly, the map boundary actors must be placed about 6000 UnrealEd units above the ground level. This tells the engine when to start playing the incoming artillery sounds.

You'll find that the map bound actors are already placed in the tutorial map, but they are at ground level. You'll find them in the bottom left and top right corners of the map. Select them and move them until they are 6000 uu's (UnrealEd units) above the ground. Since this level isn't 6000 uu's in height, we can just place them at the top of the level.

When you're done, rebuild and save the level again. Now, when you play the level, everything should be copasetic.

5.6 Conclusion

Artillery can be a key part of a Red Orchestra: Ostfront '41-45 level. A prime example of this is KonigsPlatz. Success for the soviet team depends on a commander who can work the artillery properly. Without artillery, the soviet team has almost no chance of winning that level.

In this tutorial, I've shown you how to configure artillery for either side, add a commander role, add a radio mesh, and the artillery trigger itself. I've also shown you how to properly set the map bounds actors so the sounds are properly synchronized with the artillery visuals. This is all you'll need to add artillery to any level you are working on.

Good luck.

Part 6: Destroyable Meshes Tutorial

6.1 Introduction

Battlefields are commonly considered to be unhealthy places and generally result in a sharp reduction in property values. This is due to the large amounts of high explosives that tend to get tossed around. Naturally, we'd like to be able to replicate that kind of effect in our Red Orchestra: Ostfront '41-45 levels. Well, it is possible, and I'll show you how in this tutorial.

In this tutorial I'll introduce you to the RODestroyableStaticMesh actor and show you what it can do. I'll start with a simple example then move on to a destroyable mesh that can actually be used to affect game play and alter the flow of battle in your level. Finally, I'll go over a few guidelines for the intelligent use of these meshes in your level.

For this tutorial, I'll assume you are somewhat familiar with the Red Orchestra level editor and basic Ostfront mapping topics. If not, I'd recommend reading the Quick Start Tutorial first. In fact, I'll be using the map created in that tutorial as a starting point. That map is packaged with the Quick Start tutorial so you may want to download it even if you are a veteran level designer.

6.2 A Simple Oil Barrel

I like to start out these tutorials with a simple example, just to get the feel of things. Let's start with an old classic, the exploding oil barrel. We'll create a simple barrel that will explode when shot.

Now, a little background on destroyable meshes is in order. A destroyable mesh is essentially a little sleight of hand performed by the game engine. The level starts out with one mesh. When enough damage of a particular type is applied to the mesh, it is "destroyed". A player will see the original mesh altered, either completely destroyed or simply deformed in some way. Actually, what is really happening is the engine is simply swapping the original mesh with a new one. This swap is faster than the player can see so the effect looks instantaneous. You could actually set this kind of thing up yourself using movers, but this is a lot easier.

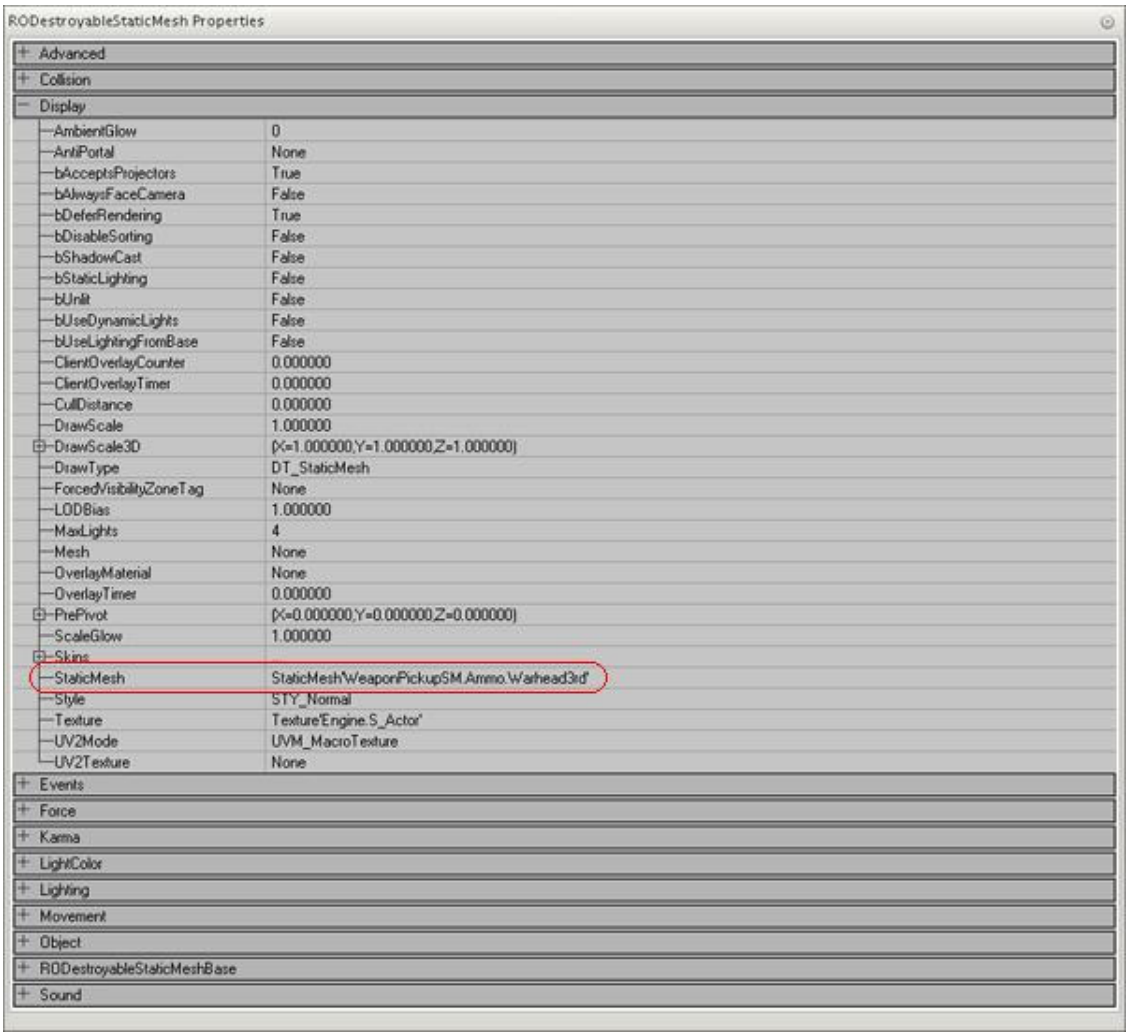
Destroyable static meshes have the following capabilities:

- They can be completely destroyed or can swap in a "damaged" mesh.
- They can be accompanied by an effect such as flying glass.
- They can throw events, which can be used to trigger other level effects.
- They can be integrated into the level overhead map.
- They can issue messages to all players when they are destroyed.
- They can be tuned so they are only affected by certain weapons.

These are powerful little tools for level designers, as we shall see.

But let's get back to our simple example. Fire up the Red Orchestra level editor and open a test level (one of your own or the one that accompanies the Quick Start Guide, it doesn't matter). Open the actor browser and find the RODestroyableStaticMesh actor. Find a spot near the center of the level and place the actor. You'll notice the actor looks like a Panzerfaust warhead. Don't worry, that's just the default form of the mesh. Let's change it to our oil barrel.

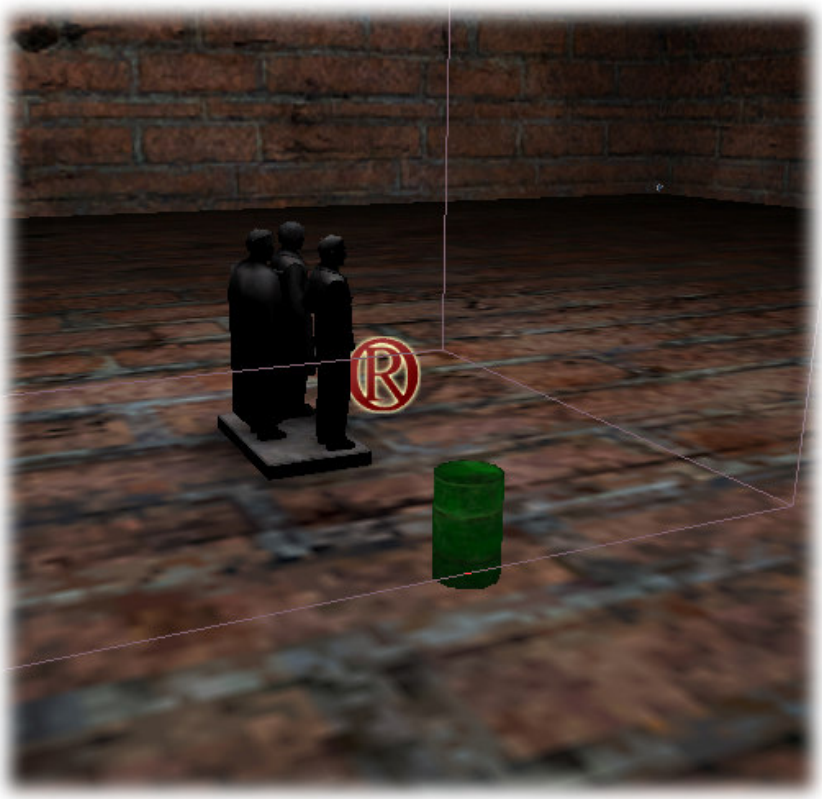
Double click on the RODestroyableStaticMesh actor to bring up its property dialog then open the Display group. You should see something like this:



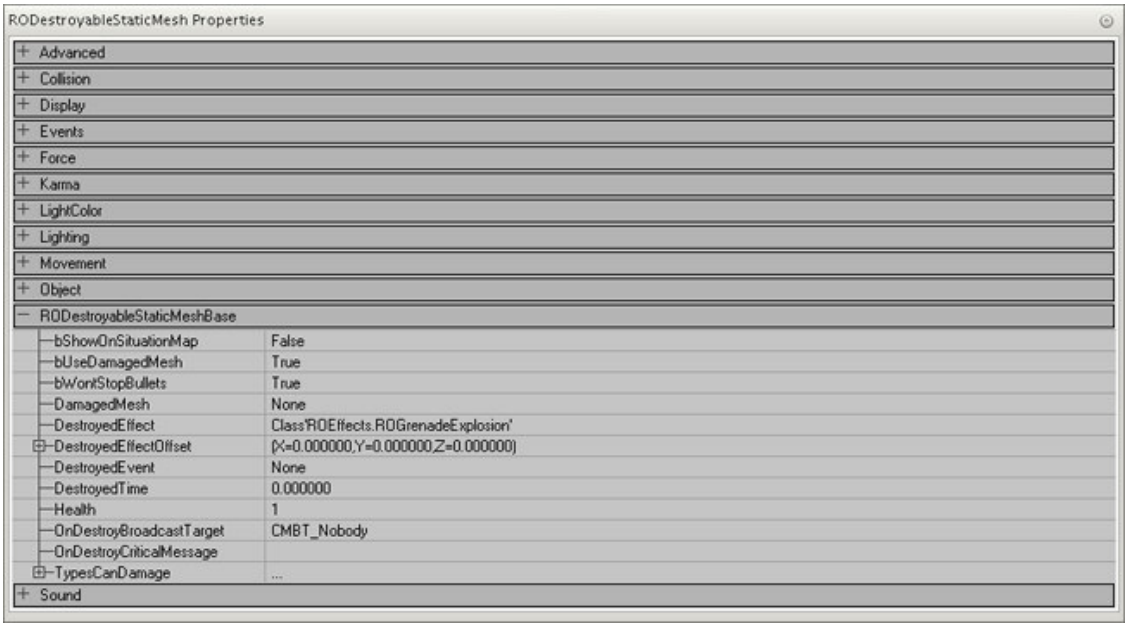
The highlighted property, StaticMesh, determines the form of the mesh in the level. We want to replace warhead with an oil barrel so we'll need to find the static mesh in the mesh browser.

Open the mesh browser then click on the "Load Package" button (the one that looks like an open folder). Find and open the IndustrySM package. You should see a bunch of barrels at the top of the list of meshes. Any one will do, but I'll use the Barrel_Green mesh. Select it and close the browser. Now go back to the RODestroyableStaticMesh properties, select the StaticMesh property and click on the "Use" button. You should see the warhead change into our barrel.

Your level should now look something like this:



Now let's set up the behavior of the barrel. Double click on the barrel again to open the properties dialog. This time we're interested in the RODestroyableStaticMeshBase group of properties. Those will look like this:



Set the following properties as shown:

- bUseDamagedMesh: False
- bWontStopBullets: False
- DestroyedEffect: ROGrenadeExplosion
- Health: 1
- TypesCanDamage: Class'ROGame.ROWeaponProjectileDamageType'

Setting that last property can be a bit tricky. All meshes can be damaged by multiple "types" of weapons (for example, bullets, grenades, grenade fragments, tank main gun rounds, etc). You'll have to configure which types of damage you want to affect the meshe you're working on. To set this property click on the "Add" button and then find ROWeaponProjectileDamageType in the drop down list. This tells the barrel we want any projectile fired from small arms to affect the barrel.

What else did we set here? We told the game we're not going to use a damaged mesh; the barrel will destroy itself entirely, leaving no remains. We want the barrel to stop bullets, so we set that property. We'll pretend that the barrel is full of some extremely unstable liquid so any bullet hit will cause it to detonate like a grenade. With that in mind, I set the effect that will be triggered when the barrel is hit to the same that is used in Red Orchestra: Ostfront '41-45 grenades. Finally, we set the health of the barrel to 1, so that any little hit will cause it to go off.

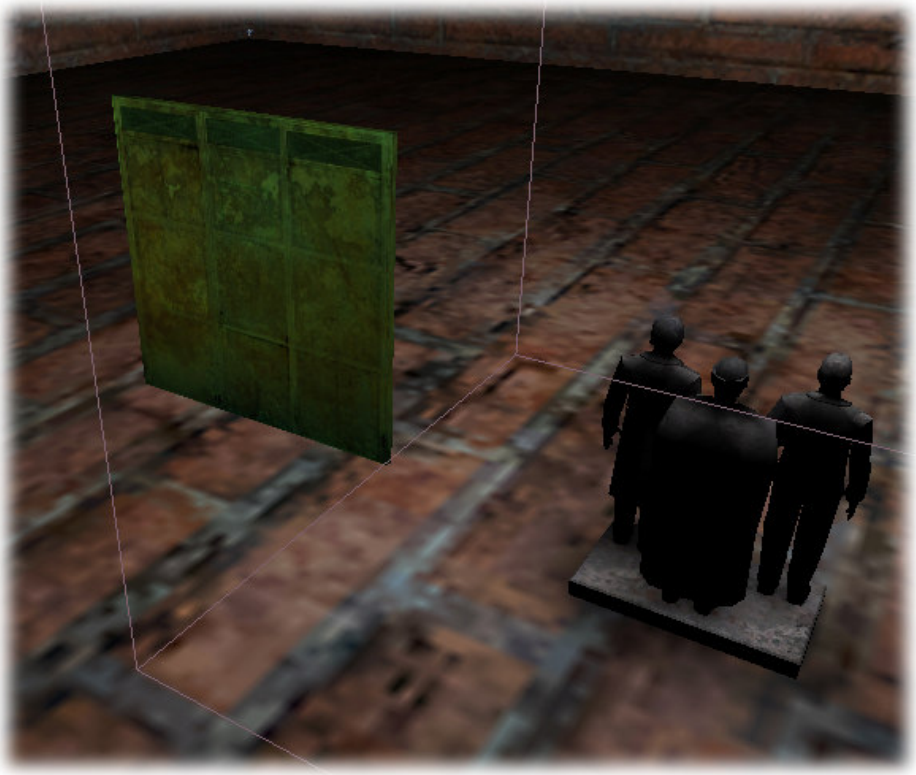
You can now rebuild and save your level (I'd suggest as RO-DestroyableMesh). In fact, go and play the level as well. If you shoot the barrel you should see if blow up in an impressive cloud of smoke.

6.3 Destroyable Meshes and Game Play

This is cool stuff, but if that were all we used destroyable meshes for it would be a waste. The real power of these meshes is their ability to affect the game play in a level. If you've played the Stalingrad Kessel level in Red Orchestra: Ostfront '41-45 then you already know what I'm talking about. The level has a number of impassable obstacles that can only be overcome by blowing them up with satchel charges. This adds a whole new dimension to the game play, as an entirely different role is required to blow the obstacles.

So how did SasQuatch set up these obstacles in that level? I'm going to show you.

Start up the Red Orchestra level editor again and load our test level. Add another RODestroyableStaticMesh somewhere near the center of the level. Again, we're going to have to replace the default mesh with the one we want. This time we're going to use one of the actual doors used in StalingradKessel. Open the mesh browser then find and load the DestructiblesSM package. I'll use destro_metaldoors01 in this tutorial. Set this as the new value in the StaticMesh property just like we did with the barrel. When you're done, you're level should look like this:



Let's move on and configure the door. Open it's property dialog and set the following properties:

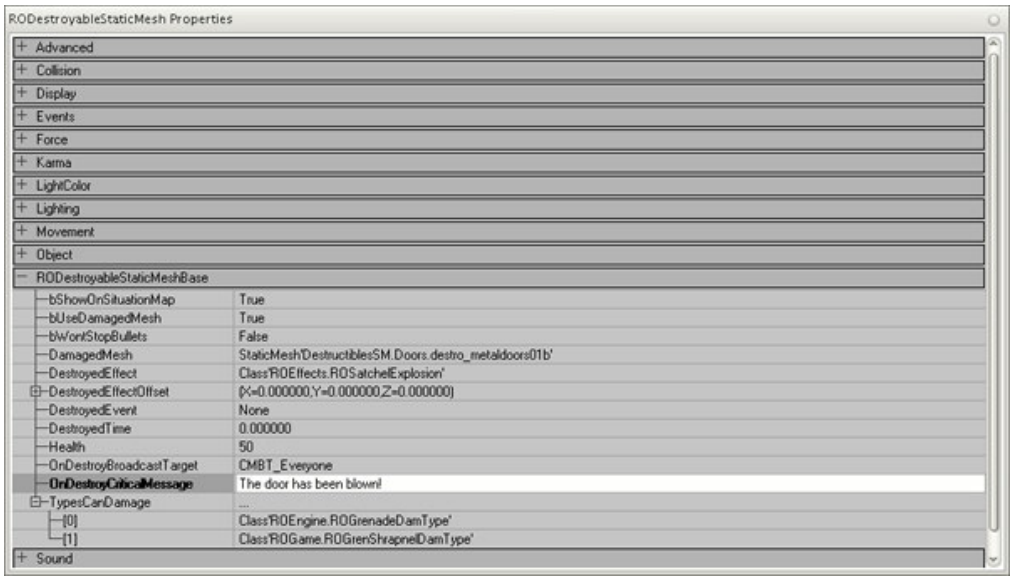
- bShowOnSituationMap: True
- bUseDamagedMesh: True
- bWontStopBullets: False
- DestroyedEffect: ROSatchelExplosion
- OnDestroyBroadcastTarget: CMBT_Everyone
- OnDestroyCriticalMessage: The door has been blown!
- Health: 50

Here we've set the door so that it will show up on the overhead map. This helps players find the obstacles that have to be destroyed. This time we want to replace the original mesh with a damaged version, so we need to set bUseDamagedMesh. I went for a really big and gaudy explosion effect on this one by using the satchel effect. Finally, we set the door health to 50 so that it's possible it might actually survive for a while.

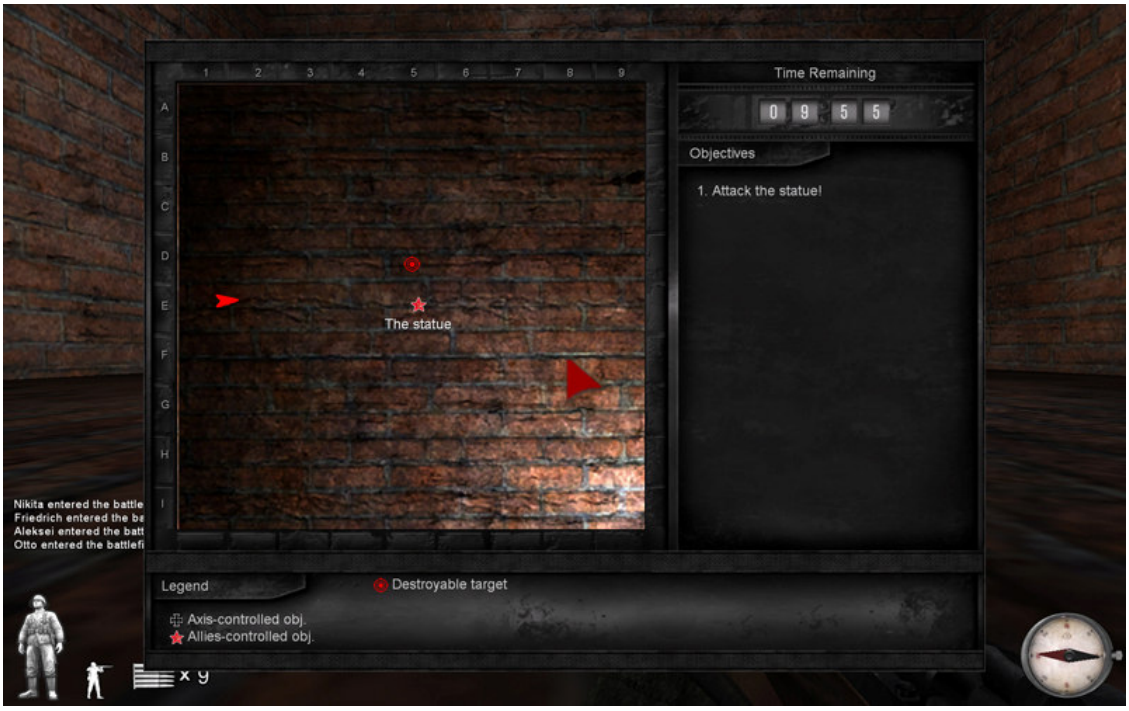
Now we need to set the damaged mesh. Go back to the static mesh browser and find the damaged version of the door you used. Select it and close the browser. Find the DamagedMesh property and use the mesh you selected.

The last thing we need to do is set what types of damage will affect the door. I think grenades make sense here, as bullets won't normally affect a steel door. There are actually two ways that grenades can cause damage: blast and shrapnel. This means we'll need to add two damage types to the TypesCanDamage property. Click the "Add" button twice to do this then search the list for ROGrenadeDamType and ROGrenShrapnelDamType.

Once you're done, the properties should look like this:



You can now rebuild, save, and play the level. Once you enter the game, open the objectives window. You should notice a red target on the map in approximately the position of the door.



Toss a grenade at the door. You'll notice a largish explosion, which will wreck the door. There should also be an announcement letting you know the door has been blown. If you take a look at the objectives screen now you'll see that the red target has become gray, noting the fact that the mesh has been destroyed.

Your grenades pack quite a punch, don't they?

You can see how valuable this would be as a level design tool. With destroyable static meshes you can control access to various parts of you map. You could even trigger the destruction of a mesh yourself based on the capture of an objective or some other event. These meshes are an extremely powerful tool and will help you create some very convincing and impressive levels.

6.4 Guidelines for the Use of Destroyable Meshes

All of this power comes at a price, however. As a general rule of thumb, destroyable meshes increase the performance load your level has to carry so use them sparingly.

Here are a few things to consider when using destroyable meshes:

1. If you're using a damaged mesh, you have to consider the impact of both meshes on your level. It's not unusual to find that the damaged version of the mesh actually has more polygons than the pristine version.

2. Don't bother using destroyable meshes where no one is likely to see them. Save decorative destroyable meshes for "in your face" moments and places.

3. If you're planning to use a destroyable mesh to control the flow of game play, then extra attention to the fighting around the mesh is advisable. If you place a destroyable mesh in a location that is easy to reach with little danger then you're missing out on a golden opportunity to create some excitement.

4. If you're creating a destroyable mesh that is supposed to explode under it's own power, then you should also consider how this would affect players. For example, you could stand right next to our barrel in our test level and shoot it and feel no ill effects. That's not very realistic. Some additional work is necessary to inflict a little damage on anyone who would do such a silly thing.

6.5 Conclusion

In this tutorial I've shown you how to use destroyable meshes both as decoration and as game play control device. I've also given you fair warning that they are a two-edged sword. They can be extremely powerful tools when used properly, but they can also really do damage to the performance of a level if used indiscriminately.

The possibilities with these tools are truly endless. I hope this tutorial has really started the gears in your head turning and you're already thinking of plenty of evil (yet fun) ways to use these in your levels. They're definitely worth it.

Good luck.

Part 7: Full Actor Reference Guide

7.1 Introduction

Red Orchestra: Ostfront '41-45 is based on the Unreal Tournament 2.5 engine. As a result, in many respects a Red Orchestra level is very similar to any other map produced in UnrealEd. However, Tripwire Interactive has extended the set of map actors supported by UnrealEd. Level designers must use many of these new actors if they wish to create working Red Orchestra levels.

This document introduces and explains the new map actors provided with Red Orchestra: Ostfront. It is intended as a reference guide for level designers who wish to create new levels for the game.

The actors described in this document do not represent the complete set provided in Red Orchestra: Ostfront '41-45. Decorative and rarely used actors such as emitters are not included.

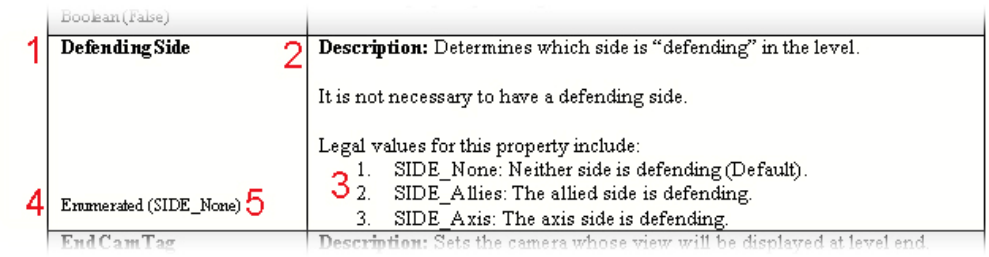
7.2 Actor Reference

7.2.1 How to Read This Document

This reference document presents the various actor properties in a tabular format. The following information is provided:

1. The property name.
2. A description of the property.
3. Where the property allows a fixed range of values, these values are identified and explained.
4. The base type of the property.
5. The property's default value, if any.

Below is a sample entry showing where each of these may be found.



In the example, the property DefendingSide is an enumerated type with 3 values. The default value for the property is SIDE_None.

7.3 ROLevelInfo

Path: Actor->Info->ROLevelnfo

Description:

This actor allows level designers to configure the setting for their map.

Every Red Orchestra: Ostfront '41-45 level must include one instance of this actor.

7.3.1 General Properties

The following properties apply to the entire level.

Property Name	Details
AlliedSatchelsPerSapper Integer (2)	**Description:** Sets the maximum number of satchel charges that can be carried by an allied combat engineer. Satchel chargers are required in order to destroy a satchel objective (See ROObjSatchel).
AxisSatchelsPerSapper Integer (2)	**Description:** Sets the maximum number of satchel charges that can be carried by an allied combat engineer. Satchel charges are required in order to destroy a satchel objective (See ROObjSatchel).
bDebugOverhead Boolean (False)	**Description:** Allows level designers to more easily configure the overhead map position.
bUseSpawnAreas Boolean (False)	**Description:** Indicates that spawn area actors (ROSpawnArea) will be used to control player spawning.
DefendingSide Enumerated (SIDE_None)	**Description:** Determines which side is "defending" in the level. It is not necessary to have a defending side. Legal values for this property include: 1. SIDE_None: Neither side is defending (Default). 2. SIDE_Allies: The allied side is defending. 3. SIDE_Axis: The axis side is defending.
EndCamTag String	**Description:** Sets the camera whose view will be displayed at level end. When the level ends, the view from the specified camera will be displayed to all players. This field should be set to the value found in the Events->Tag property of the desired camera or actor.

EntryCamTags List of strings	**Description:** Sets the cameras whose views can be displayed at level start. During the warm-up period prior to start of play, players can view selected scenes as viewed from the entry cameras. When multiple cameras are present, players will be able to cycle through the views.
MapImage String	**Description:** Sets the graphic to be used for the overhead map. During play, players have access to an overhead view of the level. The overhead map is invoked during play via the "display overhead map" button. The overhead map displays the general layout of the level as well as the player's current position. It also shows important game features such as field radios, ammo dumps, and objectives.
NumObjectiveWin Integer (0)	**Description:** Sets the number of objectives that must be captured in order for the side to "win".
OverheadOffset Enumerated (OFFSET_Zero)	**Description:** Sets the orientation of the overhead map. This property is useful when the orientation of the overhead map does not match the level. This property allows a level designer to rotate the overhead map graphic so that the map and the level are aligned. The legal values for this property include: 1. OFFSET_Zero (Default): Do not adjust the orientation of the overhead map. 2. OFFSET_90: Rotate the overhead map image 90 degrees clockwise (BUGBUG). 3. OFFSET_180: Rotate the overhead map image 180 degrees. 4. OFFSET_270: Rotate the overhead map image 270 degrees clockwise (BUGBUG).
RallyPointInterval Integer (5 seconds)	**Description:** Sets the delay, in seconds, that a commander must wait after setting a rally point before they can set another. Players in the commander role may set rally points on the overhead map. These rally points are only visible to the team and are useful for coordinating the movements of the side.
RoundDuration Integer (10 minutes)	**Description:** Sets the maximum time, in minutes, for play on this level.

StartCamTag String	**Description:** Sets the camera to use at level start. When the level begins, the view from the specified camera will be displayed on all clients. This field should contain the value of the Events->Tag property for the camera in question.

TempFahrenheit Integer (65 degrees)	**Description:** Sets the outside temperature for the level. The temperature of the level can have an affect on game play, specifically on the performance of the team machine guns. Machine guns will tend to overheat more easily on days when the weather is hotter.

7.3.2 Team Specific Properties

Each of the following properties is available for either the Allied and Axis sides.

Property Name	Details
ArtilleryBatterySize Enumerated (BAT_4_to_6)	**Description:** Sets the number of guns per battery for the side. Legal values for this property include: 1. BAT_4_to_6: A battery of 4 to 6 guns (Default). 2. BAT_8_to_12: A battery of 8 to 12 guns. 3. BAT_15: A battery of 15 guns.
ArtillerySalvoAmount Enumerated (SALVO_4_to_6)	**Description:** Sets the number of salvos in a single fire mission. Legal values for this property include: 1. SALVO_2_to_3: Two or three salvos per fire mission. 2. SALVO_4_to_6: Four to six salvos per fire mission (Default). 3. SALVO_6_to_8: Six to eight salvos per fire mission.
ArtilleryStrikeDelay Enumerated (DELAY_Short_15s)	**Description:** Sets the delay between the time the fire mission is requested and the time the first salvo is fired. This delay represents the normal passage of time as the fire request is transferred from the field radio to the artillery unit. Legal values for this property include: 1. DELAY_Short_15s: A short delay of 15 seconds (Default). 2. DELAY_Medium_30s: A moderate delay of 30 seconds. 3. DELAY_Long_1m: A long delay of 1 minute. 4. DELAY_VLong_2m: A very long delay of 2 minutes.
ArtilleryStrikeInterval Enumerated (INT_Medium_45s)	**Description:** Sets the time between fire mission salvos. This delay represents the time required to reload the guns and prepare for the next salvo. Legal values for this property include: 1. INT_Short_30s: A 30 second delay. 2. INT_Medium_45s: A 45 second delay (Default). 3. INT_Long_1m: A 1-minute delay. 4. INT_VLong_3m: A 3-minute delay.

ArtilleryStrikeLimit Integer (3)	**Description:** Sets the maximum number of fire missions available to the side.
ArtilleryStrikePattern Enumerated (STR_Normal)	**Description:** Sets the pattern for each salvo for the side. Legal values for this property include: 1. STR_Tight: A tight clustering of rounds in a salvo. 2. STR_Normal: A normal clustering of rounds in a salvo (Default). 3. STR_Loose: A loose clustering of rounds in a salvo. Rounds will cover a wide area.
Nation Enumerated	**Description:** Sets the nationality of the team's forces. Legal values for this property include: 1. NATION_SovietUnion: Side represents the Soviet Union. 2. NATION_Germany: Side represents Germany.
ReinforcementInterval Integer (30 seconds)	**Description:** Sets the delay, in seconds, between reinforcement waves for the team. Red Orchestra uses timed reinforcement waves to control player respawning. Players who are killed will wait for the next reinforcement wave to respawn. All waiting players will then respawn together. Therefore, a player who has been killed will wait no longer than the delay time specified here.
SpawnLimit Integer (150)	**Description:** Sets the maximum number of respawns for a side. Players who are killed will respawn as long as the total spawn count for the side is less than the limit. For example, if the spawn limit for a side is 100, then players can respawn a maximum of 100 times. Once the spawn limit is reached, no more reinforcements are allowed and players will not be able to respawn.
UnitInsignia String	**Description:** Sets the unit insignia for the side. Level designers can add custom graphics to represent the side. This graphic is displayed in the team selection screens when players enter a level.
UnitName String	**Description:** Sets the name of the military unit representing the side. This property allows a level designer to customize the name of the unit representing each side. The unit name appears on the level introduction screens and on the scoreboard in game.

7.4 MasterObjectiveManager

Path: Actor->MasterObjectiveManager

Description:

The MasterObjectiveManager allows level designers to control the status of multiple objectives. This is necessary in complicated maps where objectives are enabled or disabled based on player actions.

For example, a level designer may wish to create a situation in which one objective is enabled only when two other objectives are captured. A fourth objective may then depend on the status of the third, and so on. The MasterObjectiveManager gives the level designer the control they need to make it work.

The MasterObjectiveManager contains a series of simple ObjectiveManagers. Each ObjectiveManager is used to control a particular event affecting objectives. One ObjectiveManager may disable an objective while another may enable the very same objective, each based on different criteria.

The properties shown here are all members of an individual ObjectiveManager instance.

7.4.1 General Properties

Property Name	Details
ActivationStyle Enumerated (AS_Activate)	**Description:** Determines if this ObjectiveManager is enabling or disabling an objective. Legal values for this property include: 1. AS_Activate: Activate the target objective. 2. AS_Deactivate: Deactivate the target objective.
AlliesObjectivesToModify List of Integers	**Description:** A list of allied objectives whose status is to be changed. This property holds a list of the objective numbers of the allied objectives whose status will be updated when the ObjectiveManager is triggered. The objective number can be found in the ObjNum property of the relevant ROObjective actor.
AlliesRequiredObjectives List of Integers	**Description:** A list of objectives that the allied team must hold in order to trigger this ObjectiveManager. This property holds a list of objective numbers of objectives that must be held by the allied team. If the allied team does not hold these objectives, this ObjectiveManager will not be triggered. The objective number can be found in the ObjNum property of the relevant ROObjective actor.
AxisObjectivesToModify List of Integers	**Description**: A list of axis objectives whose status is to be changed. This property holds a list of the objective numbers of the allied objectives whose status will be updated when the ObjectiveManager is triggered. The objective number can be found in the ObjNum property of the relevant ROObjective actor.
AxisRequiredObjectives List of Integers	**Description:** A list of objectives that the allied team must hold in order to trigger this ObjectiveManager. This property holds a list of objective numbers of objectives that must be held by the allied team. If the allied team does not hold these objectives, this ObjectiveManager will not be triggered. The objective number can be found in the ObjNum property of the relevant ROObjective actor.

7.5 ROObjSatchel, ROObjTerritory

Path: Actor->NavigationPoint->JumpDest->JumpSpot->GameObjective->ROObjective
->ROObjSatchel
->ROObjTerritory

Description:

Together, this pair of actors allows level designers to create objectives. ROObjSatchel creates satchel objectives that are "captured" by detonating a satchel charge within the objective control area.

The ROObjTerritory objective creates objectives that represent an area on the map. These objectives are captured when players occupy the control area for a specified amount of time.

7.5.1 General Properties

Property Name	Details
AlliesEvent String	**Description:** Sets an event to be thrown if the allied side captures the objective.
AlliesObjectivePriority Integer (0)	**Description:** Sets the relative priority for the objective for the allied side. This priority is used by bots to determine which objective to attack.
AttackerDescription String	**Description:** Sets the text that appears on the level objectives list for the attacking side. This text will only appear to players on the attacking side.
AxisEvent String	**Description:** Sets an event to be thrown if the axis side captures the objective.
AxisObjectivePriority Integer (0)	**Description:** Sets the relative priority for the objective for the axis side. This priority is used by bots to determine which objective to attack.
bRequired Boolean (True)	**Description:** Indicates that the objective is required in order for the attacker to win the level. Not all objectives must be captured in order for a side to win.
DefenderDescription String	**Description:** Sets the text that appears in the level objectives list for the defending side. This text will only appear to players on the defending side.

InitialObjState Enumerated (OBJ_Neutral)	**Description:** Sets the initial state of the objective. Objectives can be set to any of the following states: 1. OBJ_Neutral 2. OBJ_Axis 3. OBJ_Allies
MapX	**Description:** Sets the x position of the objective in the overhead map. This property is no longer used.
MapY	**Description:** Sets the y position of the objective in the overhead map. This property is no longer used.
objName String	**Description:** Sets the name of the objective. This name appears on the overhead map.
ObjNum Integer (0)	**Description:** Sets the number of the objective. The objective number is used by other actors to refer to this objective (See MasterObjectiveManager). Red Orchestra objective actors cannot share the same objective number.
Radius Integer (1024 Unrealed units)	**Description:** Sets the radius of a circle that defines the control area for the objective. There are two methods of setting the control area for an objective: by radius or by volume. Setting the radius is the simplest method by the control area fixed as a circle. Using a volume allows level designers to create flexible shapes for the control zone.
VolumeTag String	**Description:** Sets the tag of a volume that defines the control zone for the objective. There are two methods of setting the control area for an objective: by radius or by volume. Setting the radius is the simplest method by the control area fixed as a circle. Using a volume allows level designers to create flexible shapes for the control zone. This property should be set to the string that is entered in the Events->Tag property of the control zone volume.

7.6 ROAmmoPickup

Path: Actor->Pickup->Ammo->ROAmmoPickup

Description:

The ROAmmoPickup family of actors was originally intended to allow players to acquire additional ammunition during play. The introduction of ROAmmoSupplyVolume makes these actors much less useful and they should generally not be used in a level.

There is one exception to this rule and that is panzerfausts. In battles that pit a primarily infantry based German force against a mechanized Soviet force, additional panzerfaust pickups may be critical for balanced game play. In such cases, it is perfectly acceptable to use ROPanzerFaustPickup.

7.6.1 ROAmmoPickup Properties

Property Name	Description
DropLifeTime Float (45 seconds)	**Description:** Sets the time, in seconds, that the ammunition will remain in play after being dropped. Once this time has expired, the ammunition will be removed from the level.
TouchMessage String	**Description:** Sets the message that is displayed to the player when the ammunition is picked up.

7.6.2 ROPanzerFaustPickup Properties

Property Name	Description
WeaponType String	**Description:** Sets the type of weapon for the pickup. This property should not be changed.

7.7 RODestroyableStaticMesh

Path: Actor->RODestroyableStaticMeshBase->RODestroyableStaticMesh

Description:

Red Orchestra: Ostfront '41-45 provides level designers with the ability to add static meshes that can be destroyed during play. This is useful for several reasons:

1. To prevent access to certain parts of a map until the mesh is destroyed.
2. To create special effects and decorations, such as breaking glass.
3. To create objectives.

Destroyable meshes appear like any other static mesh in game. On top of this, destroyable meshes are also linked to a destruction event. Each destroyable mesh can be configured to be damaged by a specific group of weapons. Once sufficient damage is done to a destroyable mesh, it triggers a destruction sequence. At that time, the original mesh may be swapped with a "damaged" mesh. The destruction sequence can also trigger other effects like emitters.

To the players' eyes, the destruction of a mesh is instantaneous. Properly done, the effect is totally believable and can produce impressive results.

7.7.1 General Properties

Property Name	Description
bShowOnSituationMap Boolean (False)	**Description:** Show the location of the destroyable mesh on the level overhead map. This may be desirable when the destroyable mesh plays a key role in the flow of the level. For example, if the destroyable mesh represents doors that need to be blown, then it is likely important to show the location of these doors on the map.
bUseDamagedMesh Boolean (True)	**Description:** Indicates that the destroyable mesh will be replaced by another, damaged mesh. It is not always desirable to have a damage mesh. For example, a level designer may want to simulate the complete destruction of the original mesh (say, breaking glass).
bWontStopBullets Boolean (False)	**Description:** Indicates whether the original mesh blocks bullets or not. In certain instances, a level designer may not want the destroyable mesh to block bullets. For example, if the destroyable mesh represents a wooden door, it may be desirable for bullets to pass through the object.

DamagedMesh String	**Description:** Sets the name of the mesh to be used when the original is destroyed. The damaged mesh is only used if the *bUseDamagedMesh* flag is set to True.
DestroyedEffect String	**Description:** Sets the emitter to be used when the mesh is destroyed. This property should be set to a specific instance of the emitter.
DestroyedEffectOffset Vector of Integer	**Description:** Sets the offset, in UnrealEd units, where the destroyed effect emitter will be spawned. The offset in question is relative to the original destroyable mesh. The offset includes X, Y, and Z components.
DestroyedEvent String	**Description:** Sets the tag of the event to be thrown when the mesh is destroyed.
DestroyedTime Float	**Description:** The time at which the mesh was destroyed. Do not change this property.
Health Integer (0)	**Description:** Sets the health of the original mesh. When the health of the mesh reaches zero, the destruction sequence is triggers. At that time, the damaged mesh, if used, will be swapped with the original. Any destruction effect will be initiated and the destruction event will be thrown.
OnDestroyBroadcastTarget Enumerated (CMBT_Nobody)	**Description:** Sets a target for any message triggered by the destruction of this mesh. Legal values for this property include: 1. CMBT_Nobody (Default): No message is sent. 2. CMBT_Everyone: The message is transmitted to everyone. 3. CMBT_Teammates: Only players on the same team as the player that destroys the mesh will see the message. 4. CMBT_Enemies: Only players on the opposing team from the player that destroys the mesh will see the message. 5. CMBT_Instigator: Only the player that destroys the mesh will see the message.

OnDestroyCriticalMessage String	**Description:** Sets a critical message that will be transmitted when the mesh is destroyed. Level designers have the option of setting a message that will be transmitted when a mesh is destroyed. This message will be displayed in the corner of the players HUD. Who receives this message depends on the value set in the OnDestroyBroadcastTarget property.
TypesCanDamage List of Strings	**Description:** Sets the list of weapons that can damage the mesh. It is highly unlikely that a level designer will want to allow all weapons to affect a destroyable mesh. This property allows the level designer to tailor the weapons that can destroy the mesh in question.

7.8 ROMapBounds

Path: Actor->ROMapBounds
->ROMapBoundsNE
->ROMapBoundsSW

Description:

All Red Orchestra: Ostfront '41-45 levels support an overhead map, which displays a top down view of the battlefield. The map displays the battle objectives as well as other important points such as ammo dumps, radios, rally points, and artillery strikes. Perhaps most importantly, the map also displays the players' position on the battlefield.

For level designers, the overhead map can be a bit of a challenge to create. The map is usually a screenshot of the actual battlefield taken in game. Screen shots like this make excellent maps, but they can be tricky to align correctly. The ROMapBounds actors help make this job easier.

The map boundary actors actually serve two purposes:

1. The let the game engine know the boundaries of the map. The enables the important points of interest on the map to be displayed in the correct positions.
2. The set the ground level of the map. This enables artillery to work correctly by synchronizing the sounds of incoming rounds with the actual battlefield detonations.

The map boundary actors have no properties. They are simply placed at the proper locations on the map in the UnrealEd editor. The ROMapBoundsSW actor should be placed at the extreme "southwest" corner of the map, usually the bottom left. The ROMapBoundsNE actor is then placed in the opposite corner. If the actors are placed properly, all landmarks on the battlefield will then appear in their proper positions in game.

As noted earlier, the boundary actors also affect the performance of artillery. The sound of incoming rounds is noticeable long before the round actually impacts the ground. The game must know the base level of the ground in order to calculate when to begin playing the sound of an incoming round. For this reason, the map bounds actors should be placed 6000 UnrealEd units above the base ground level.

7.9 ROMineField

Path: Actor->ROMineField

Description:

Minefields are commonly employed in many conflicts and the eastern front was no exception. Minefields can be useful to level designers in many ways:

1. As a normal minefield, intended to restrict access to parts of the battlefield.
2. As a barrier to player movement in places where terrain or some other obstacle is not available.

Currently, only two types of mines are modeled in Red Orchestra:

1. The "Schu" mine. This was a small wooden box about 6 inches square and a favored mine of the German armed forces.
2. The PMD6 anti-personnel mine. These are the favored anti-personnel mine of the Soviet forces. These mines can still be found in use today.

This actor has been supplemented with a new volume type, ROMineVolume. At first glance these actors may seem redundant but each fullfills a different role. The ROMineVolume actor should be used in places where the level designer wants to first warn players they are entering a forbidden game area. The ROMineField actor behaves more like the real thing: it is hidden, deadly, and doesn't differentiate between friend and foe.

7.9.1 General Properties

Property Name	Description
MineClass Enumerated (ROPMD6Mine)	**Description:** Sets the class of mine included in the minefield. Legal values for this property are: 1. ROMine 2. ROPMD6Mine: The Soviet PMD6 anti-personnel mine (Default). 3. ROSchuMine: The German Schu mine. 4. ROSMine
NumMines Integer (8)	**Description:** Sets the number of mines to be found in the minefield. The number of mines will affect the likelihood of a player encountering one when traversing the field.
XWidth Integer (256 UnrealEd units)	**Description:** Sets the size of the minefield in the X-axis, in UnrealEd units.
YWidth Integer (256 UnrealEd units)	**Description:** Sets the size of the minefield in the Y-axis, in UnrealEd units.

7.10 RORoleInfo

Path: Actor->RORoleInfo

Description:

Not all soldiers have the same job on the battlefield and this is no different in Red Orchestra: Ostfront '41-45. Red Orchestra defines a number of different roles that can be selected by players.

The RORoleInfo family of actors allows level designers to define:

1. The branch or service arm that the soldiers belong to, for each side.
2. The types of soldiers are present on the battlefield, for each side.
3. How many of each type is available.
4. Cosmetic details such as seasonal uniform types, for each side.

7.10.1 Soldier Roles

The following table shows the roles that are available for use:

Role	Description
Assault	A basic infantry role. Usually armed with a sub-machine gun.
Combat engineer	A soldier trained in demolitions.
Anti-tank	A soldier armed with an anti-tank weapon, either an anti-tank rifle or a panzerfaust.
Machine gun crew	A soldier armed with a machine gun.
Rifleman	A basic infantry role. Usually armed with a bolt action.
Semi-automatic rifleman	Another basic infantry role. Usually armed with a semi-automatic rifle.
Sniper	A specialized rifleman armed with a bolt action or semi-automatic rifle equipped with a telescopic lens.
Leader	The unit leader. Armed with either a rifle or sub-machine gun. Also equipped with binoculars. Can direct artillery fire.
Tanker	A tank crewman. Armed with a pistol.
Tank commander	A tank crewman and leader. Armed with a pistol and equipped with binoculars. Can direct artillery fire.

7.10.2 German Soldiers

German soldiers are organized into two main branches, or service arms:

1. The Heer, or army, which includes Wehrmacht and Luftwaffe troops.
2. The Waffen SS.

Both service branches offer a variety of uniform types. The different types do not affect gameplay, but do alter the appearance of soldiers in game. These sub-groupings of roles allow level designers to customize the look of the soldiers to fit the time of year.

Branch	Role Group	Description
Heer	ROGE_Greatcoat_Heer	Cold weather uniforms.
	ROGE_Splinter_Heer	Uniforms for lesser-known service branches such as Luftwaffe field divisions.
	ROGE_Standard_Heer	Regular summer uniforms.
Waffen SS	ROGE_FallCamo_SS	Fall camouflage uniforms.
	ROGE_SpringCamo_SS	Spring camouflage uniforms.
	ROGE_Standard_SS	Standard SS uniforms.

7.10.3 Soviet Soldiers

Soviet soldiers are also organized into service arms. These are:

1. The NKVD, or People's Commissariat for Internal Affairs.
2. The RKKA, or Red Army
3. The RKKF, Soviet naval troops (Worker and Peasants Red Fleet).

As with the Germans, all service branches offer a variety of uniform types. The different types do not affect gameplay, but they do alter the appearance of soldiers in game. These sub-groupings of roles allow level designers to customize the look of the soldiers to fit the time of year.

Branch	Role Group	Description
NKVD	ROSU_FallCamo_NKVD	Fall camouflage uniforms.
	ROSU_SpringCamo_NKVD	Spring camouflage uniforms.
RKKA	ROSU_Greatcoat_RKKA	Cold weather uniforms.
	ROSU_Spring_RKKA	Spring camouflage uniforms.
	ROSU_Winter_RKKA	Winter uniforms.
RKKF	ROSU_TankCommander_RKKF	Tank commander
7.10.3.1	ROSU_TankerRKKF	Tank crewman
7.10.3.2	ROSU_Standard_RKKF	Standard uniforms

7.10.4 General Properties

Property Name	Description
bCarriesMGAmmo Boolean (False)	**Description**: Indicates that the role carries additional ammunition for the machine gun crews. In Red Orchestra: Ostfront '41-45, players can receive additional points for re-supplying machine gun crews. If this property is set to true, the role carries extra ammunition and may re-supply machine guns.
GivenItems List of Strings	**Description:** Sets a list of additional items that can be carried by the role.
Grenades List of Items	**Description:** Sets a list of grenades carried by the role. Red Orchestra: Ostfront '41-45 allows each role to carry up to 3 types of grenades (through this is not recommended). All roles carry two of the standard grenade (for the side in question), by default. Each grenade type is defined by 3 additional properties: 1. Amount: The number of grenades 2. AssociatedAttachment: Not used. 3. Item: The grenade class itself.
limit Integer (0)	**Description:** Sets the maximum number of players that can select this role. Level designers can set a cap on the number of players that may select a particular role. If this property is set to 0, then there is no limit and any number of players may select the role.
PrimaryWeapons List of Items	**Description:** Sets the primary weapons for the role. As with grenades, level designers can set up to 3 primary weapons for a role. If more than one primary weapon is set, then the player may select from the weapons when they enter the level. Each weapon is defined by 3 additional parameters: 1. Amount: The number of ammunition clips for the weapon. 2. AssociatedAttachment: The object associated with the weapons ammunition. 3. Item: The weapon class itself.

PrimaryWeaponType Enumerated	**Description:** Sets the type of the primary weapon. Legal values for this property include: 1. WT_Rifle 2. WT_SMG 3. WT_LMG 4. WT_Sniper 5. WT_SemiAuto 6. WT_Assault
SecondaryWeapons List of Items	**Description:** Sets the secondary weapons for the role. Level designers can set up to 3 secondary weapons for a role. If more than one secondary weapon is set, then the player may select from the weapons when they enter the level. Each weapon is defined by 3 additional parameters: 4. Amount: The number of ammunition clips for the weapon. 5. AssociatedAttachment: The object associated with the weapons ammunition. Item: The weapon class itself.
Side Enumerated (SIDE_Allies)	**Description:** Sets the team for the role. Legal values for this property include: 1. SIDE_Allies (default) 2. SIDE_Axis Do not change this property.

7.11 ROSpawnArea

Path: Actor->ROSpawnArea

Description:

The ROSpawnArea actor defines an area on the map in which players "spawn" or enter the game area.

Players enter the game only through spawn areas. A level may have multiple spawn areas but only one spawn area is active at any given time.

Spawns may be come active or inactive depending on events on the battlefield. For example, the capture of a particular objective may change the spawn that is used by a side.

Two factors determine if a spawn will be used: its precedence and its required objectives. Each spawn has a precedence number. Spawns with higher precedence will be picked for use before spawns with a lower precedence. Each spawn may also depend on certain objectives being in friendly hands. If these required objectives are not held, the spawn cannot be used.

So, the game determines which spawn to use by finding the actor with the highest precedence among those whose objective requirements have been met.

7.11.1 General Properties

Property Name	Description
Allied Precedence Integer (0)	**Description:** Sets the precedence of the spawn in the allied set of spawns. Precedence determines which spawn is currently used by a side. If more than one spawn is created for a given side, then the one with the highest precedence is used.
AlliesRequiredObjectives List of Integer	**Description:** Sets the list of objectives that must be in allied hands in order for the current spawn to be used. The use of a spawn may depend on particular objectives being in friendly hands. This property contains a list of objective numbers that must be held in order for it to be used. These values should be the same as values found in the ObjNum property of the ROObjective actors in use. Together with precedence, this property determines which spawn will be currently used.
Axis Precedence Integer (0)	**Description:** Sets the precedence of the spawn in the axis set of spawns. Precedence determines which spawn is currently used by a side. If more than one spawn is created for a given side, then the one with the highest precedence is used.
AxisRequiredObjectives List of Integer	**Description:** Sets the list of objectives that must be in axis hands in order for the current spawn to be used. The use of a spawn may depend on particular objectives being in friendly hands. This property contains a list of objective numbers that must be held in order for it to be used. These values should be the same as values found in the ObjNum property of the ROObjective actors in use. Together with precedence, this property determines which spawn will be currently used.
bAlliesSpawn Boolean (False)	**Description:** Indicates that the spawn is to be used by the allied team only.
bAxisSpawn Boolean (False)	**Description:** Indicates that the spawn is to be used by the axis team only.
bInitiallyActive Boolean (True)	**Description:** Indicates that the spawn is to be active at the start of play.

bTankCrewSpawnArea Boolean (False)	**Description:** Indicates that the spawn area is to be used by tank crew roles. Often, level designers will want to spawn infantry and tanks in separate areas. Tank crews should spawn with their tanks, not with the infantry. This property allows level designers to create spawn areas for tank crew separate from the rest of the infantry roles.
SpawnProtectionTime Integer (0 seconds)	**Description:** Sets the time, in seconds, that players should be protected from damage. When a player spawns, they can be protected from damage from players on the other side. This gives incoming players a chance to get their bearings before being killed. If a newly spawned player fires a weapon, any spawn protection is removed.
TeamMustLoseAllRequired Enumerated (SPN_Neutral)	**Description:** Sets the conditions upon which the spawn area will be disabled. As previously noted, a spawn may require multiple objectives be held in order for the area to be used. This property determines if the loss of any one of the required objectives disables the spawn. For example, an allied spawn can only be used if objectives 1 and 2 are held. This property allows a level designer to determine if the spawn will be disabled if either objective 1 or 2 are lost, or if the spawn continues to be used until both are taken. Legal values for this property are: 1. SPN_Neutral: The loss of any required objective will disable the spawn (Default). 2. SPN_Allied: The spawn will be disabled only when all required allied objectives are lost. 3. SPN_Axis: The spawn will be disabled only when all required axis objectives are lost.
VolumeTag String	**Description:** Sets the tag of a volume that will be used to define the spawn area boundaries. Level designers can used a volume to define the area covered by a spawn. The volume is used in conjunction with spawn protection. Players inside this area will be protected if spawn protection is enabled. Once players leave the area, spawn protection is removed. The value in this property should be set to the string found in the Events->Tag property of the associated volume.

7.12 ROVehicleFactory

Path: Actor->SVehicleFactory->ROVehicleFactory

Description:

The fighting on the eastern front was characterized by battles involving large numbers of armored fighting vehicles. Using the ROVehicleFactory family of actors, level designers can add many of the more common vehicles to their maps.

Red Orchestra: Ostfront 41-45 includes the following vehicles:

- German vehicles:
  - PzKw IIIF
  - PzKw IVF1
  - PzKw IVF2
  - PzKw V (Panther)
  - PzKw VI (Tiger 1)
  - Stug III
  - Sdkfz 251 Halftrack
- Soviet vehicles:
  - BA64 armored car
  - T34/76
  - T34/85
  - T60
  - IS2
  - SU76
  - KV-1

In addition, most of these vehicles come with additional "snow" skins that can be used in levels that take place during winter conditions. Snow versions are supported by their own, separate map actors.

7.12.1 General Properties

Property Name	Description
bAllowOpposingForceCapture Boolean (False)	**Description**: Indicates that the tank factory may be captured by the opposing team. The captured tanks may be used by their new owners.
bAllowVehicleRespawn Boolean (True)	**Description:** Indicates that vehicles will respawn at this location.
bUseBuildEffects Boolean (False)	**Description:** Indicates that an effect will be used when new vehicles are spawned. This property should almost always be left as False.
bUseSpawnAreas Boolean (False)	**Description:** Indicates that spawn areas defined by ROSpawnArea actors will be used.

RespawnTime Float (1 seconds)	**Description:** The delay, in seconds, that will pass before a new vehicle is respawned to replace one that is destroyed.
TeamNum Enumerated (NEUTRAL)	**Description:** Sets the team that owns the vehicle factory. Legal values for this property include: 1. NEUTRAL (default) 2. AXIS 3. ALLIED
VehicleSpawnLimit Integer (3)	**Description:** Sets the number of vehicles that will be produced by the factory. Once the factor reaches the number of vehicles specified here, it will no longer respawn new tanks.

7.13 ROArtilleryTrigger

Path: Actor->Triggers->ROArtilleryTrigger

Description:

Artillery played a significant role in most battles on the eastern front and so level designers will naturally want to include it in many of their levels.

To successfully introduce artillery into a level, the following is required:

1. At least one commander role slot must be included.
2. The ROMapBounds actors must be properly placed in the level.
3. At least on ROArtilleryTrigger must be present.

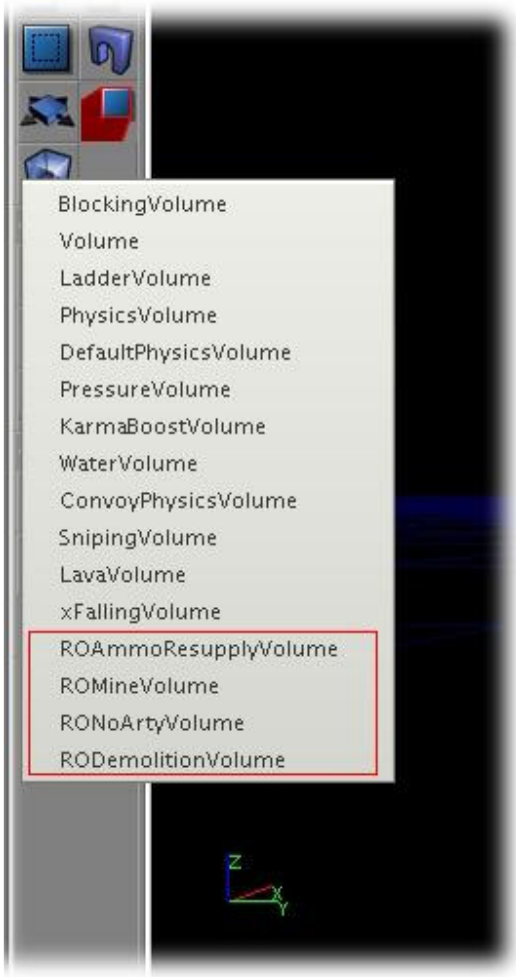
The ROArtilleryTrigger represents the radio used to contact headquarters in order to request artillery support. Generally, level designers will place a static mesh in the level to represent the radio and then place a ROArtilleryTrigger nearby. Players in the commander role can then use the radio (via the trigger) to initiate artillery strikes.

7.13.1 General Properties

Property Name	Description
Message String	**Description:** Sets the message to display to players when they enter the active radius of the artillery trigger. The message helps inform players that they are close enough to the radio to initiate an artillery strike. The standard message used in Red Orchestra maps is "Use the radio to request artillery support from HQ."
TeamCanUse Enumerated (AT_Both)	**Description:** Sets the team that can use this particular radio. In some instances, level designers will want to allow both sides to use a radio. In other cases, level designers will want to restrict access to a radio to a particular side. Legal values for this property are: 1. AT_Both (Default) 2. AT_Axis 3. AT_Allied

7.14 Red Orchestra UnrealEd Extensions

Red Orchestra: Ostfront '41-45 also provides some specialized extensions to the standard UnrealEd interface. Currently, these extensions all take the form of additional volume types. These volumes can be accessed in the brush panel of UnrealEd (shown below).



As you can see from the picture to the left, Red Orchestra provides the following new volume types:

1. ROAmmoResupplyVolume
2. RONoArtyVolume
3. RODemolitionVolume
4. ROMineVolume

The ROAmmoResupplyVolume does pretty much what you expect. It defines areas in a level where players can go to resupply their ammunition stocks.

A level designer may want to use artillery in a level but, for game balance purposes, may not want artillery to be targeted on all parts of the map. The RONoArtyVolume defines areas in which artillery may not be targeted. Volumes associated with spawn areas are automatically no artillery areas as well.

The RODemolitionVolume lets level designers specify areas in which satchel charges have special effects. These may be objectives, or simply areas in which demolitions will clear obstacles and open new avenues for movement.

Finally, the ROMineVolume allows level designers to create areas that can be used as barriers to player movement. These areas manifest themselves as mine fields in game, but they are really intended to prevent players from moving through specific areas of the map. This makes them especially useful for protecting spawn areas and limiting the play area in a level.

7.14.1 ROAmmoResupplyVolume Properties

Property Name	Description
bUsesSpawnAreas Boolean (False)	**Description:** Associates the ammunition resupply area with an existing ROSpawnArea volume. If this property is set to True, and the Events->Tag property of the ROAmmoResupplyVolume is set to the tag of a ROSpawnArea actor, the ammo resupply volume will then become enabled or disabled along with the spawn area.
Team Enumerated (OWNER_Neutral)	**Description:** Sets the owner of the resupply area. Only players this side will be able to use this resupply area. Legal values for this property include: 1. OWNER_Neutral (default). 2. OWNER_Axis 3. OWNER_Allied If the property is set to OWNER_Neutral, either side may use the resupply area.
UpdateTime Float (4.0 seconds)	**Description:** Sets the interval (in seconds) between periods when players can use the area to resupply ammunition.
ResupplyType Enumerated (RT_Players)	**Description:** Sets the type of resupply provided by this volume. Legal values for this property include: 1. RT_Players: Only players are resupplied (Default). 2. RT_Vehicles: Only vehicles are resupplied. 3. RT_All: All combatants are resupplied.

7.14.2 RONoArtyVolume Properties

Property Name	Description
SpawnAreaTag String	**Description:** Associates the no artillery volume with an existing ROSpawnArea volume. If this property is set to the Events->Tag property of a ROSpawnArea actor, the no artillery volume will then be enabled or disabled along with the spawn area. Since ROSpawnAreas now automatically includes artillery protection, this property should not be used.

7.14.3 RODemolitionVolume Properties

Property Name	Description
bIsObjective Boolean (False)	**Description:** Indicates that the volume represents an objective.
DetonationEvent String	**Description:** Sets the event that will be thrown when a satchel charge is detonated in the volume.

7.14.4 ROMineVolume Properties

Property Name	Description
KillTime Float (2 seconds)	**Description:** Sets the time, in seconds, that must pass before a player is "killed" by mines.
WarnInterval Float (3 seconds)	**Description**: Sets the interval, in seconds, between the warnings issued to a player that has trespassed into the mine volume.
WarningMessage String	**Description:** Sets the message that will be issued to players that trespass into the mine volume.
MineKillStyle Enumerated (KS_All)	**Description:** Sets the team that will be affected by the mine volume. Legal values for this property include: 1. KS_All: Both teams will be affected by the mine volume. Any player entering the mine volume will trigger its effects. 2. KS_Axis: Only players on the axis team will trigger the mine volume's effects. 3. KS_Allied: Only players on the allied team will trigger the mine volume's effects.
bUsesSpawnAreas Boolean (False)	**Description:** Associates the ammunition resupply area with an existing ROSpawnArea volume. If this property is set to True, and the Events->Tag property of the ROMineVolume is set to the tag of a ROSpawnArea actor, the mine volume will then become enabled or disabled along with the spawn area. This is particularly useful when dealing with advancing spawn areas.