

Part 5: The Finishing Touch

In this last tutorial we'll finish the score system that keeps track of all important parameters of your game such as player health, ammo and time. To display all this information we will need to make bars and counters in the interface. We also need to make sure that all interface elements align right on various resolutions. When this is done we have to setup a sequence, the main script, that handles the overall game structure (starting, ending and resetting). Now you'll hopefully understand why we had to create all those groups, initial conditions and why you don't want to make a mess of your scripts! When everything is looking good and we've done some debugging, it's time to export it to the .VMO format. We'll take a quick look at export settings and then we're done!

The score system needs a system that handles explosions and it needs to send messages to our main script when the player is dead, the time limit has been reached or when you've completed the game. The trick here is to setup thresholds and conditions to keep the parameters doing what you want them to do. For the interface we'll be working with the Display Progression Bar BB and the Bitmap Text Display BB. To make all elements align right we'll use the Get Screen Proportional Pos BB. Creating the overall game-sequence (startup/play/game-over/restart) isn't very hard. It mostly consists of sending a lot of messages at the right time to all the different scripts in your level. You will notice that in this tutorial we assume that you know how to connect and use certain building blocks and parameter operations.

When you've followed all the steps accurately you now should have a fully working 3D webgame. Don't forget to put it online for the world to see!

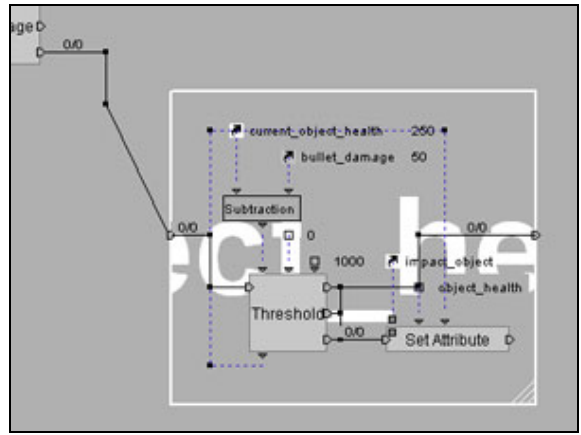
You can continue with your own project or open tutorial_part_5_start.CMO.



Estimated time to complete:
- 6 hours



Source material included (Virtools)
- effect 3D frames
- interface elements
- additional material (textures, sounds etc.)



1 OBJECT HEALTH Now that we can kill the enemies, it's a good idea to determine what other objects you can destroy in our composition.

- Create a new group containing all vehicles you want to destroy (for our composition we've left out the big busses and trucks, they're too big to blow up). Call it "destructable_group". Assign all objects in this group an "object_health" attribute. Set the object_health for the vans to 500, for the rest to 300 (use the Attribute Manager). Select all objects in the group and set initial conditions on them. Set IC on the group as well.

- Back in the "score" script, create a new message for the Switch On Message BB ("object_health_down"). Create a Threshold BB (Min:0, Max: 1000) a Set Attribute BB and a Subtraction paramOp. We want to get the current_object_health and decrease it by the bullet_damage and then put it back on the impact_object, like we did for the character_health. You need to use shortcuts to parameters in the "fire" script, inside the bullet impact BG. When you have opened the "fire" script, you can use the search function (Ctrl+F) to quickly find the necessary parameters.

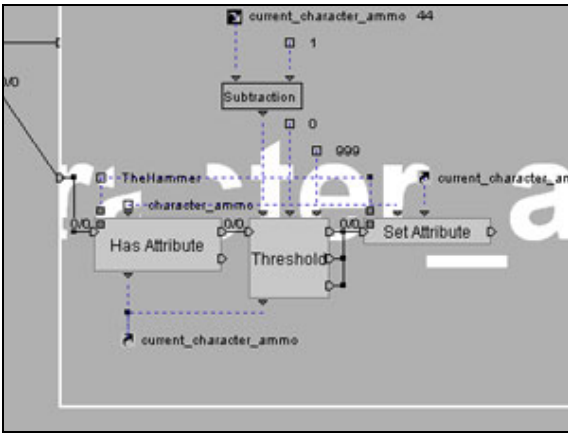
- Back in the "score" script, create a BG "object_health" around these BBs, with one bOut (connected to the X<Min bOut of the Threshold BB). Make sure you have all bOuts of the Threshold BB connected to the Set Attribute BB.

- While we have the "fire" script open, (the bullet impact BG), we need to insert one small test before we can activate the Mark bIn of the Mark System BB. If the impact_object is in the destructable_group, the Lifespan and Fading Time pIns of the Mark System BB need to change (so that the marks disappear much faster, we don't want marks floating around after our vehicles have exploded). For this, use an Is In Group BB and two Parameter Selector BBs. If the impact_object is not in the destructable_group, the Lifespan and Fading Time can remain 10s and 3s, when the object is in the group you can set them to 500ms and 100ms.

Test your script. Although nothing happens when you hit a vehicle, its object_health attribute should have decreased.

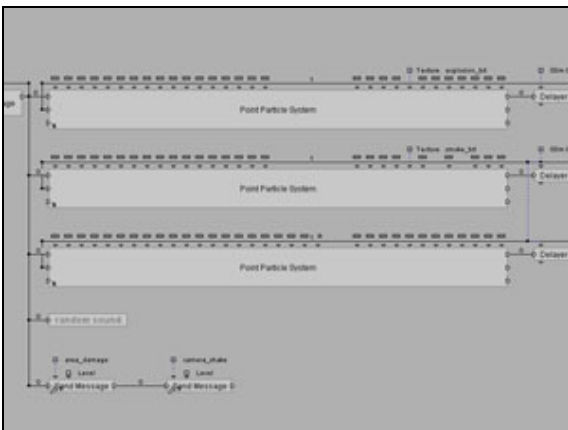


Tip: You can disable the enemies Level Script (so it isn't Activated At Scene Start) so you won't be bothered by all those enemies when your testing this part of your script.



2 AMMO TEST We still need to add a counter for the player's ammo. Again, we'll use a Threshold BB, a Set Attribute BB and a Subtraction paramOp, but this time we need to start by retrieving the Hammer bodyparts current character_ammo attribute by using a Has Attribute BB. Subtract the current_ammo value by 1 before putting it back into the bodypart. Set the Min and Max pIns of the Threshold to 0 and 999. Create a BG around it and call it "character_ammo". Connect its bIn to the character_ammo_down bOut of the Switch On Message BB. Notice in the screenshot that for some parameters we've use shortcuts, others we've connected directly. Create your scripts in a way that is most clear for you!

- When the player is out of ammo, he shouldn't be able to fire bullets. There should be a clicking sound as well. Go to the fire Level Script. Behind the Wait Message ("fire") BB, make some room and insert a Has Attribute BB (Target: TheHammer bodypart, Attribute: character_ammo), a Test BB (to test if this value is Greater Than 0) and a Play Sound Instance BB. For this BB we need to import a new sound called hammer_noammo.mp3. In its setup, uncheck Streamed, check Point and specify the dummy_character as the Attached Object. Set the Min and Max perception distances to 5 and 100. Create a BG with two bOuts around it called "ammo_test" loop the lower bOut (connected to the False bOut of the Test BB) directly back to the Wait Message BB, connect the upper bOut (connected to the True bOut of the Test BB) to the get_new_bullet BGs bIn.



3 BLOWING STUFF UP When a vehicle loses all its health it should of course explode! So, we need a nice explosion! Import explosion_frame.NMO into your scene. This file contains a 3D Frame, two sound effects and two textures. Let's take a closer look.

- Take a look at the setup of the 3D Frame. Notice that its direction (z-axis) is pointing upwards (you can see this in the viewport as well). This is because when we position our explosion somewhere on the ground, we

want most of our particles going upwards (the standard direction for the particles is over the frames z-axis).

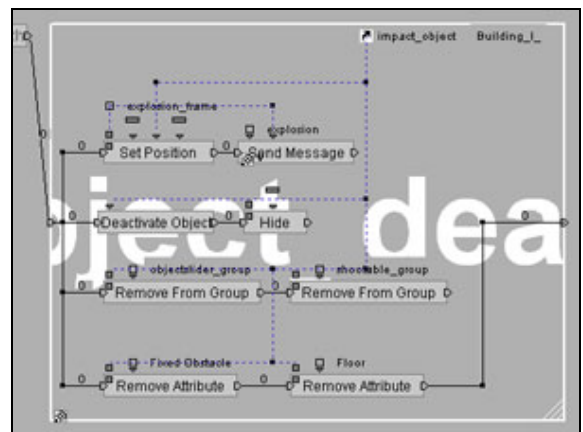
- Now, take a look at the setup for the two sound effects (Misc_explosion_1 and Misc_explosion_2). Again, Streamed is unchecked (you can't play instances of streamed sounds), Point is checked (because we want to position these sounds in the 3D environment), and set the Min/Max perception distances to 3 and 4 (making them loud). There is no Attached Object specified because we are going to need these sound for other things as well. If you take a look at the two textures, explosion_txt and smoke_txt, notice that the first one is composed of 16 frames of animation. If we want it to play correctly, we need to specify this in our particle system.

- Open up the script attached to the frame in the Schematic view. Notice that there are three Point Particle System BBs. The first one provides the explosion. When you take a look at the lower parameters of this BB, you can see that the Texture Frame Count (16) multiplied by the Texture Speed (125ms) is exactly the Lifespan (2s) of the particles (so that all of the animation is displayed). In the settings ("S") of the BB, notice that the Trail Particle Count has been set to 5 to give it more volume. Take a look at the parameters of the second Point Particle System BB (used to produce some white smoke). We want these particles to have a more horizontal movement (but the standard direction of the particles is upwards). So we have changed the Yaw and Pitch Variance to 90 degrees. The third Point Particle System BB produces line-particles to create some sparks. Check the settings ("S") of this BB. You can see that the Particle Type has been set to Line. Also, the Infinite Plane deflector and the Gravity interactor have been checked (remember that we have already created that particle_frame with the corresponding attributes?). These particles also have trails.

- Notice that we have added some Delayer BBs to make sure the particle systems stop when we want them to. There's also a BG "random_sound" that provides a nice explosion sound. Underneath that there are two Send Message BBs. One for area damage and one to create a camera shake (we'll handle that later). You have to set "Level" as the Destination of both BBs! The whole thing is activated by a looped Wait Message BB ("explosion").



Tip: For optimization purposes, always uncheck all unused features of Particle Systems in the settings of the BB.



4 EXPLOSION CONTINUED Back in our "score" script, we need to write the behavior that triggers this explosion and that deactivates the exploding object. We'll create a new BG called "object_death" behind the "object_health" BG.

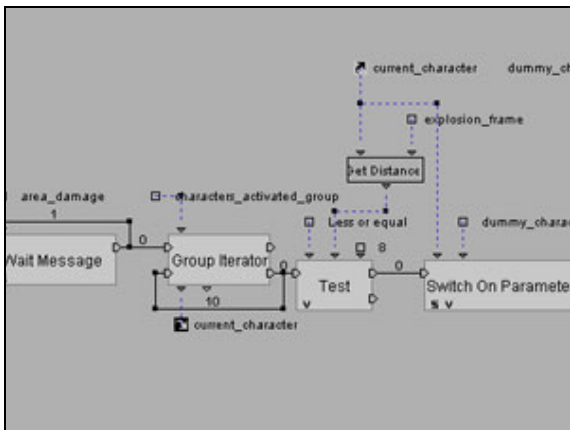
- The first thing we need to do is to set the position of the explosion_frame onto the impact_object. Then we need to send the explosion message to the frame. We need to do a lot of things to the impact object. We have to deactivate it, hide it, remove it from the objectslider_group and the shootable_group and we need to remove floor and fixed obstacle attributes. All these things are important because we don't want any "ghost" objects in our scene.

- Note: we could have placed all BBs behind each other, creating one long row. The result would have been the same. This way created a better screenshot!

- Important: before we test our script, we need to set IC on the objectslider_group (if you haven't done so already!). Otherwise objects would be permanently removed from this group!

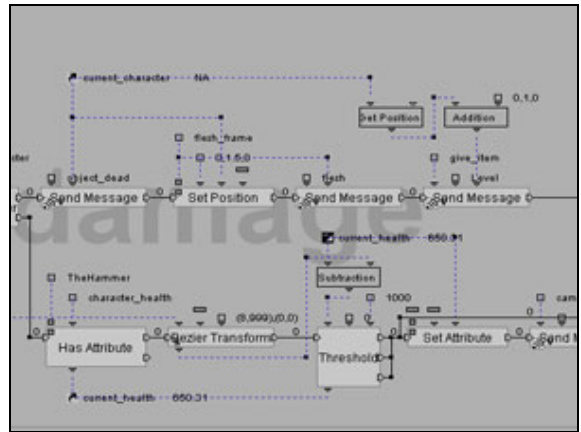
OK, it's time to blow some stuff up! Feel free to tweak the settings of the particle system until you are completely satisfied! For a future game, you could even create wreck-meshes for your vehicles, so you wouldn't have to hide them!

Tip: If you hold shift when you move a BB, a paramOp or a parameter, you will make a duplicate or copy of it.



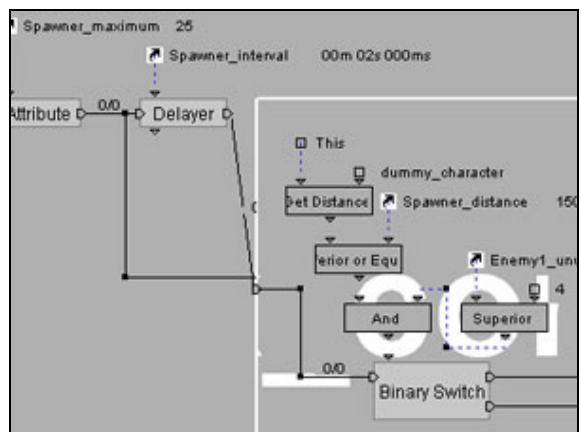
5 AREA DAMAGE PART ONE It's nice that we have things exploding around us, but if I would stand next to an explosion in the real world, I'd be dead (or I'd lose a lot of health!). Next, we'll create a system that handles this, using a Group Iterator BB.

- In the score script, underneath that Switch On Message BB, add a looped Wait Message BB ("area_damage"). Behind that, create a Group Iterator BB (Group:characters_activated_group). It's very important that the dummy_character is in this group, if it isn't already (otherwise the player won't be checked correctly). Loop the Group Iterator BB (you don't have to include all following BBs in the loop to make this work). Set a link delay of 10 for this loop (so our enemies have some time to die before we check the next one). Now test if the distance between the current character (create a new parameter underneath the Group Iterator BB) and the explosion_frame is Less or equal to 8 meters. If this is True, use a Switch On Parameter BB to check if this character is the player (dummy_character).



6 AREA DAMAGE PART TWO If the current_character is not the dummy_character (so that the "none" bOut of the Switch On Parameter is activated) then: we need to send the "object_dead" message to this character. We also have to position the flesh_frame on its location (Position: (0, 1.5, 0) with the current_character as the Referential). We also have to send the flesh message to the Level. The last thing we need to do is to send the "give_item" message to the Level, so that we get an item for our troubles. Don't forget that you have to construct a vector parameter input for this BB. Use paramOps to get the position of the current_character, added by (0, 1, 0). Insert this into the new vector pIn of the Send Message BB.

- If the current_character is the dummy_character, we need to retrieve the character_health attribute from TheHammer bodypart and store it into a new temporary local float parameter called "current_health". Behind this we can add a Bezier Transform BB to change the distance (from the current_character to the explosion_frame, you can use the same paramOp if you want) into a damage amount. Insert the distance into the x pIn of the Bezier Transform BB. Specify (8, 999)(0,0) as the Boundaries Rectangle. We are going to subtract this damage amount from the current_health using a Threshold BB and a Subtraction paramOp like we did a couple of times before. Then we put the current_health attribute back into TheHammer bodypart using the Set Attribute BB. You also want to be notified when health is subtracted so let's add another Send Message BB (Message: camerafilter_blood, Destination: Level). Create a BG around this part of your script called "area_damage". Connect its bIn to Start. Create two bOuts for it, the top one connected to the upper Send Message BB ("give_item"), the lower one connected to x<Min bOut of the Threshold BB.




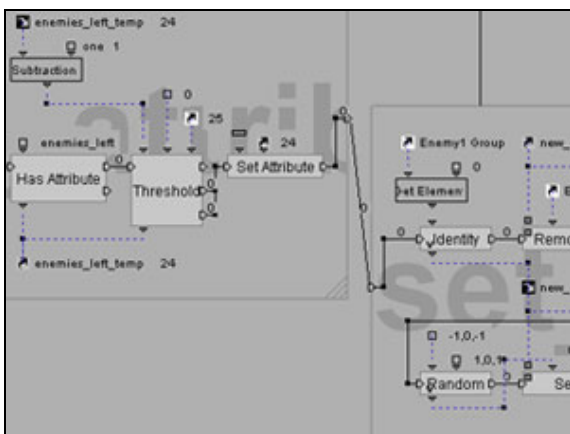
7 ENEMY SPAWNER For our game, we want the enemies to appear to be coming from the black vans parked in our environment. Each van should

spawn about 25 enemies. For this to happen, certain conditions need to be met: 1. The van must be active (not be destroyed) 2. The van must not have spawned 25 enemies already. 3. There must be enemies available (they mustn't all be in use). 4. The player must be in a certain range of the van. If all of these conditions are true, the van may take an enemy out of the enemy1_group, randomly orientate him (so they won't come out all the same) and then activate him.

- In the enemies Level Script, we need to create a little something that keeps track of how many enemies are still available for deployment and store this value in the already created "Enemy1_unused" parameter in this script. Use a looped Identity BB, connected to a Get Count paramOp (specify the enemy1_group as the group). This outputs an integer but it will be converted to a float automatically. Insert the pOut of the Identity BB into the Enemy1_unused parameter. Create a BG around this called "enemy1_unused".

- Now, in the attribute manager, create a new float attribute in our Object Properties category called "enemies_left". Now create a script "spawner" on one of the vans. First we need to give the van the right enemies_left starting value. Create a Set Attribute BB (Attribute: enemies_left) and insert a shortcut to the spawner_maximum parameter (found in the enemies Level Script) into the value pIn. Behind that, create a Delayer BB (use a shortcut to "spawner_interval"). Add a Binary Switch BB. Using paramOps, this BB must test if the distance between This and the dummy_character is inferior or equal to the "Spawner_distance" AND if the Enemy1_unused value is superior to 4 (a lower value could cause a bug...). Create a BG called "check_conditions" around this BB, with one bOut for the True and one bOut for the False bOuts of the Binary Switch BB. Loop the lower bOut of the BG back to the Delayer BB.

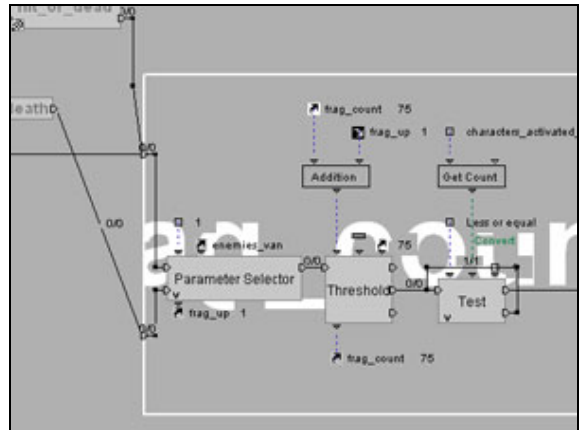
 **Tip:** Most of the time it's a good idea to use attributes instead of local parameters (send by a message or something) when these values are specific for a certain (copied) object. Notice that setting the initial attribute using a BB in your script gives you the possibility to easily tweak your settings using a local parameter value somewhere in your script.



8 SPAWNER CONTINUED Now, let's use the Has Attribute BB (create a new local parameter underneath this one called "enemies_left_temp"), Threshold BB (Min:0, Max: spawner_max), Subtraction paramOp and Set Attribute BB combination again to decrease the newly created "enemies_left_temp" by one and then to put it back into the enemies_left attribute. Don't connect the x<0 bOut of the Threshold BB to the Set Attribute BB (if the van has spawned its 25 it needs to do nothing). Create a BG around these BBs called "check_attribute".

- Behind this we'll create another BG called "set_enemy". Use an Identity BB and a Get Element paramOp to get element 0 from the Enemy1_group. Insert this into a new character parameter called "new_enemy". Now we need to do a couple of things to this enemy: Remove it from the Enemy1_group, Set its Position (Position: (-1, 1.5, -0.5) Ref: This), Set a Random (between (-1,0,-1) and (1,0,1) Orientation for it using a Random BB and a Set Orientation BB. Then we have to Add the enemy to the characters_activated_group and finally we have to Activate it (using the Activate Object BB: set the Reset pln to False, the other two to True, otherwise it would be set back to its initial position). Loop the bOut of this BG back to the Delayer BB.

- In the Level Manager, select this script and use the right click menu to copy and paste this script onto the other vans as well. Now make sure that Enemy1 and its AI script aren't activated at scene start by clicking the "A" icons behind their name (Set IC on the character!). Do this for the Vans as well. Test your script! Be sure to test if the spawning stops when the vans are destroyed!



9 FRAG COUNTER The player has won our game when he has killed all (4 x 25 = 100) enemies. So, we could count how many times an enemy is killed using our hit_or_dead BG inside our score script. However, if one or more vans is destroyed, you can never reach a score of 100. If a van is destroyed, we need to add the enemies that are left "inside" the van to the score.

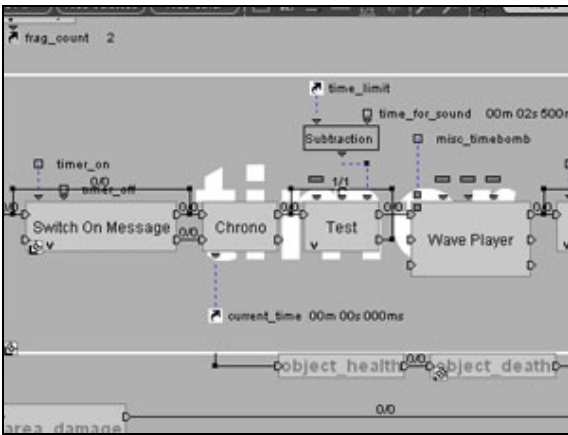
- First, create two bOuts for our hit_or_dead BG. Inside it, connect the bOut of the UPPER Send Message BB (give_item) to the LOWER bOut of the BG. Delete the lower Set Position BB and Send Message BB (the ones for the flesh_frame on the hammers position). Connect the lower bOut of the lower Switch On Parameter BB directly to the UPPER bOut of the BG (we're going to create some game-over behavior elsewhere). This way each time an enemy is killed, the lower bOut of the BG will be activated. When the Hammer is killed, the upper bOut will be activated.

- Now inside the object_death BG, insert a Has Attribute BB before the bOut of the BG is activated. We need to check how many enemies the impact_object has left (if any)! Create a new parameter underneath the Has Attribute BB (Attribute: enemies_left) called "enemies_van" (and connect it, of course!). We'll use this parameter later on.

- Create a new local float parameter in the top part of your script (where you keep the rest of them). Call it "frag_count". Create another one called "frag_limit", set its value to "75". Create an Identity BB in the top part of your script and connect it so Start to make sure the frag_count is set to 0 each time the script is activated.

- Now create a new Parameter Selector BB next to the object_death BG. Specify the value "1" as the first pIn and a shortcut to the enemies_van value as the second pIn. Create a new local parameter called "frag_up" underneath the BB. Create a Threshold BB (Min:0, Max: frag_limit) and an Addition paramOp. Use it to add frag_up to frag_count in the same way as we did for all those attributes. Only connect the x>max bOut to a new Test BB. Use it to test if the characters_activated_group has more than 1 element (that last one is the player!) using a Get Count paramOp. The frag_limit may be reached, but you only win if you kill all the enemies that are still walking around!

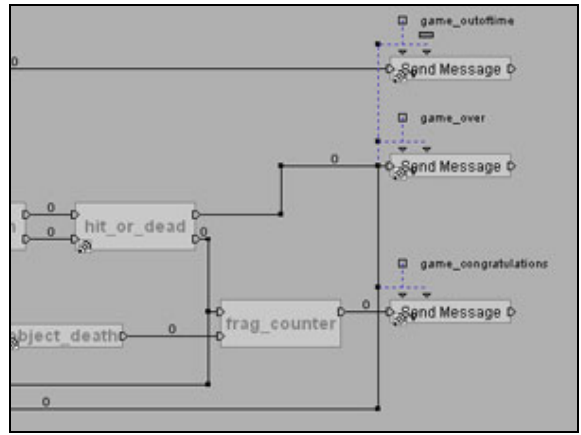
Loop the False bOut and connect the True bOut to the bOut of the BG you have to create for these last BBs. Call it "frag_counter". Now, the lower bOut of the hit_or_dead BG and the upper one of the area_damage BG need to be connected to the upper bIn of this BG. The object_death BGs bOut needs to be connected to the lower bIn of the BG!



10 TIMER We also want to put a time limit on our game. Create two new local time parameter in the top part of your script, one called "current_time" and one called "time_limit" (value: 2 minutes).

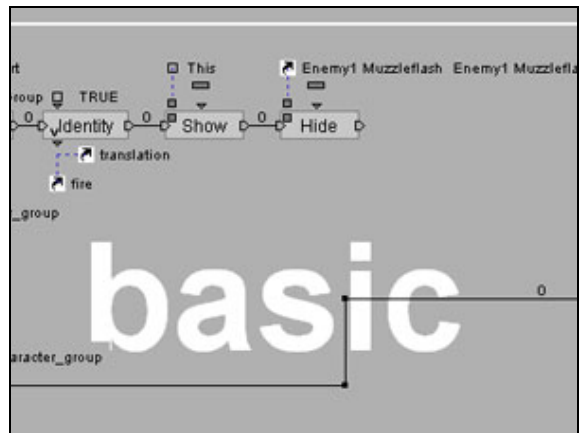
- Create a Switch On Message BB ("timer_on" and "timer_off"). Create a Chrono BB. Paste a shortcut to "current_time" underneath its pOut (that's right:connect it! I bet you've already guessed how to connect the Switch On Message BB to the Chrono BB as well!).

- When your time is almost up, we want a sound to start playing. Import Misc_timebomb.mp3 (uncheck Streamed in its setup). Behind the Chrono BB, insert a Test BB (loop the False bOut back into its bIn!) to test if the current_time is Greater or equal to the time_limit minus 2,5 seconds (the time for the sound effect to reach booooo!). Use a paramOp to do this. If this condition is True, use a Wave Player BB to play the sound. When the Exit On bOut of this Wave Player BB is activated, use another Test BB to test if the current_time is Greater or equal than the time_limit. Create a nice BG around it called "timer", connecting the True bOut of the last Test BB to its bOut. For now, maybe it's a good idea to connect the bIn of this BG directly to the Chrono BBs bIn (so that it's activated automatically).

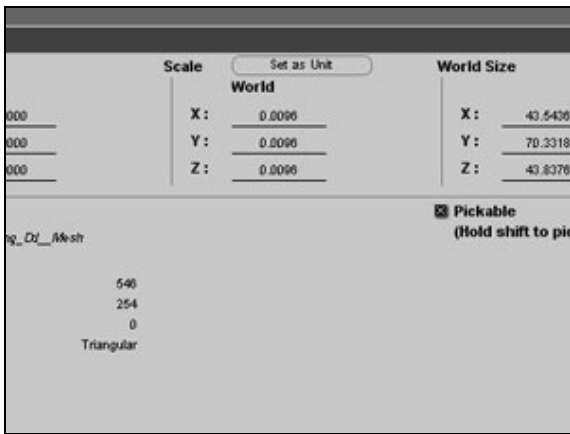


11 GAME MESSAGES Our game can end in three different ways: the player can be out of time, he can be dead, or he wins the game. We've already created the right tests for all these conditions, so we can set up three Send Message BBs to send the right messages to a script that handles the main game states (we still need to create that one).

- Create the three Send Message BBs underneath each other in the right part of your script ("game_outoftime", "game_over" and "game_congratulations"). Set the Destination of the all three BBs to Level. Connect the bOut of the "timer" BG to the "game_outoftime" BB, the upper bOut of the "hit_or_dead" BG and the lower bOut of the "area_damage" BG to the "game_over" BB and the bOut of the "frag_counter" BG to the "game_congratulations" BB. Before we can save our progress up to now, it may be a good idea to do a small bughunt first!

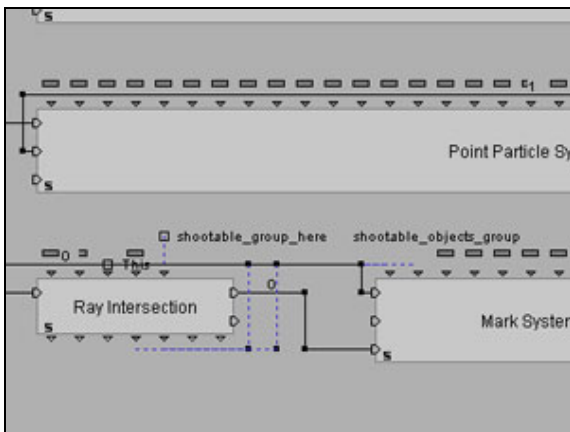


12 ANNOYING MUZZLEFLASH You may have noticed that sometimes an enemy muzzle flash is already visible when an enemy is spawned. Hopefully this is fixed by adding a Hide BB in the "basic" BG of the AI script. Minor bug.

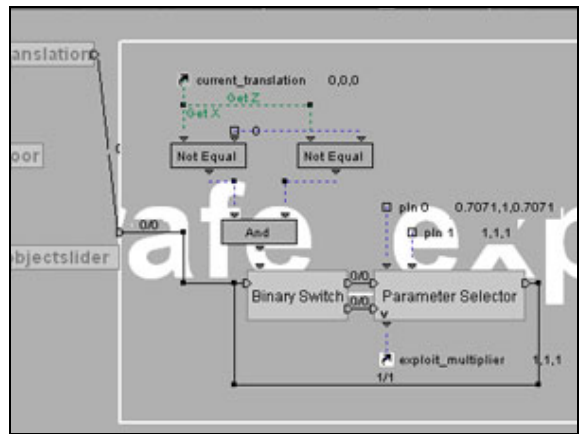


13 OBJECT SLIDER TROUBLES When you see characters moving through buildings or other objects, most of the time this is because the objects scale is not (1, 1, 1). Check all your objects (using the 3D Object Setup) and click Set As Unit for each object that does not have (1, 1, 1) as its world scale. Don't forget to check if the size of the object is still correct and to re-set ICs! You can also use the right-click menu in the Level Manager to "SetAsUnit" objects directly. Choose Actions > 3DEntities > SetAsUnit.

Tip: A wrong world scale is usually caused by scaling of your object/poly in your modeling package. Usually the modeling package provides ways to reset pivots, world scale, bounding boxes etc (in 3dsmax you can use the Reset Xform Utility).

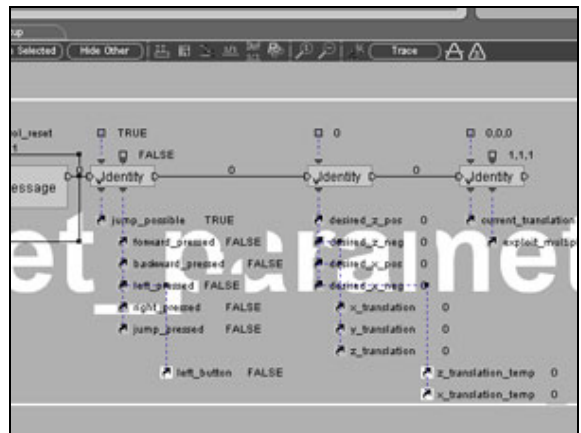


14 FLYING BLOOD STAINS As you may remember, we use a Ray Intersection BB inside the script of the flesh_frame, to create a stain where an enemy explodes. Sometimes this ray seems to hit the bodypart of the enemy so it creates a stain flying in mid-air. This is because the group we use for the Ray Intersection BB, the shootable_group, contains the enemy bodyparts. To fix this, create a new group (for example called "shootable_objects_group") that contains all of the same objects, minus both character bodyparts. Use this group for the Ray Intersection BB. Let's hope this will fix it!



15 STRAFE EXPLOIT At this moment, you actually run faster while you are strafing diagonally (when you press forward and sideways at the same time). This is because you have a z translation of 4 and a sideways translation that is also 4, resulting in a total diagonal translation of $\sqrt{32}$ (= 5,6568...). So then you run faster! OK, so when you move diagonally (when both the x and the z component of our current_translation are Not Equal to 0), we must multiply the translation by 0,7071 ($4/\sqrt{32}$). Above this paramOp we multiply the current_translation with a new vector called "exploit_multiplier". This value is (1, 1, 1) (when the movement is in one direction), or (0,7071, 1, 0,7071) when movement in multiple directions is detected. To do this, add a "strafe_exploit" BG as shown above.

We've done this by inserting a Multiplier paramOp where the current_translation is inserted in the Translate BB (in the "translation" BG in the control script of the dummy_character).

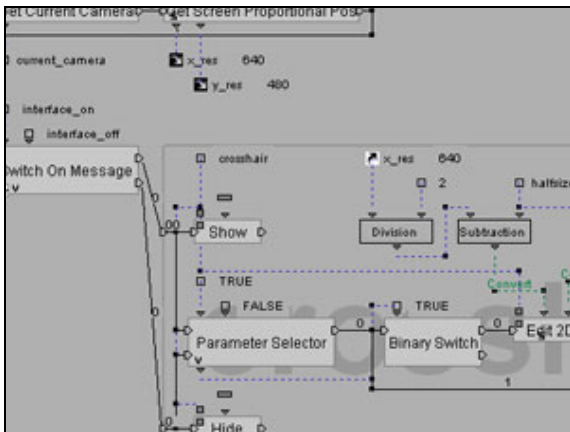


16 RESETTING MOVEMENT PARAMETERS While we're in the control script, let's make some other adjustments. First we need to create something that resets all those parameters that keep wrong values when you stop your composition while moving around. Paste shortcuts to all parameters that might need resetting in the top part of your script. Examples of parameters that need resetting: jump_possible, forward_pressed, desired_z_pos, current_translation and all similar parameters. Maybe it's a good idea to include a shortcut to left_button (located in the "temp" Level Script and also the exploit_multiplier created in the last step. The best BB for setting an initial value is still the Identity BB. You can construct new pIns by using the right click menu. Don't forget to jump_possible needs to be set to "true" and exploit_multiplier needs to be set to (1, 1, 1). In our case, it may even be a good idea to have a looped Wait Message BB ("control_reset") so we can reset these values when we want to. Create a nice BG around all this and call it "reset_parameters". Connect it to Start.

Connect the bIn of the BG to the first Identity BB so that it is activated when the script is activated as well.

- The last thing we're going to change here is in the "movement" BG. Inside the "z_translation" and the "x_translation" BGs there are Variation BBs that need to have their Reset bIns activated when the "movement_keys_on" message is activated (or else your movement can get stuck). You may need to create new bIns for the z_translation and x_translation BGs to make this work.

We've saved our progress up to now as: **tutorial_part_5_halfway.**



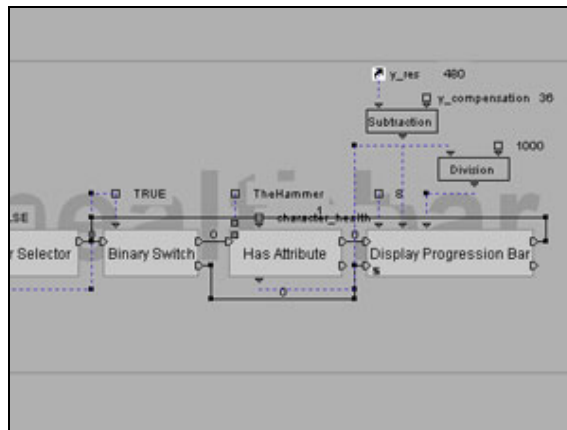
17 INTERFACE Now we'll work on the interface for a bit. We still need some graphics for a healthbar, an ammo-counter and something to keep track of time. All these things need to be positioned correctly and you must be able to turn them on and off. From your 3D Entities Resources, import the following files: interface_score.NMO (a 2D Frame with material and texture), interface_timer.NMO (a 2D Frame as well), healthbar.NMO (containing two 2D Sprites) and my_font.NMO (another 2D Sprite that defines our font).

- Go to the interface Level Script. Above the camerafilter_blood BG, insert a Get Current Camera BB (also add a new camera parameter underneath its pOut called "current_camera") and a Get Screen Proportional Position BB (set 100% for both Pins and add two integer parameters, x_res and y_res, for the two pOuts). These two BBs need to be looped and always active. This way we also have the current resolution and current camera when we need them!

- Before we do anything else, let's create a Switch On Message BB ("interface_on" and "interface_off"). To the right of that BB, add a Show BB and a Hide BB (both targeting our crosshair 2D Sprite). Also add a Parameter Selector BB ("True" and "False" Booleans) attached to a Binary Switch BB (you've guessed it: connect the pOut of the Parameter Selector BB to the Condition of the Binary Switch BB. Connect its True bOut to a new Edit 2D Entity BB (in the settings of this BB, uncheck everything except Position). Again, set the crosshair as the target. Now use parameter operations to calculate the right x and y pIns for this. We need to divide the x_res by 2, and then subtract half the crosshairs size (16) from the result (because Virtools uses the top left corner of 2D Entities for positioning). Do the same for the y pIn. Loop the Bout back to the bIn of the Binary Switch. If the condition becomes false, the loop will be broken. Create a BG ("crosshair") around it, with two bIns (so you can connect them to the bOut of the Switch On Message BB). Hide the crosshair in the Level Manager and set its IC. You need to add a Send Message BB (Message: "interface_on", Destination: Level) to your temp Level Script so it is activated properly.



Tip: Sometimes it's easier to use the "Use homogenous coordinates" checkbox in the setup for 2D Entities to keep them positioned correctly. This also scales the 2D Entities so that they occupy the same amount of screen-space on higher resolutions (this can result in unwanted texture-scaling).



18 INTERFACE CONTINUED We'll use the same technique to position the interface_score and interface_timer. You can copy the "crosshair" BG and remove the paramOps above the Edit 2D Entity BB. Interface_score needs to be positioned always 8 pixels from the left side of the screen (so the x pIn can be a static parameter with value "8"). The y position must be always 120 pixels from the bottom of the screen (this means y_res minus 120). Interface_timer needs to be positioned top center. So the x pIn needs to be the half of the x_res minus 32, the y pIn can have a static value "8",

- For the healthbar we're going to do something different: underneath the timer BG, create a Parameter Selector BB and a Binary Switch BB (much like you did before) but you can forget about the Show BB and the Hide BB. Behind that, create a Has Attribute BB and a Display Progression Bar BB. Use the Has Attribute to get the character_health from the Hammer bodypart, divide it by 1000 and insert it into the Progression percentage of the Display Progression Bar BB. Set the x location to "8" and the y location to y_res minus 36. In the settings of the BB, specify the correct 2D Sprites. Loop it back to the Binary Switch BB. Be sure to connect the False bOut to the Off bIn of the Display Progression Bar BB. Create a new BG around it called "healthbar".

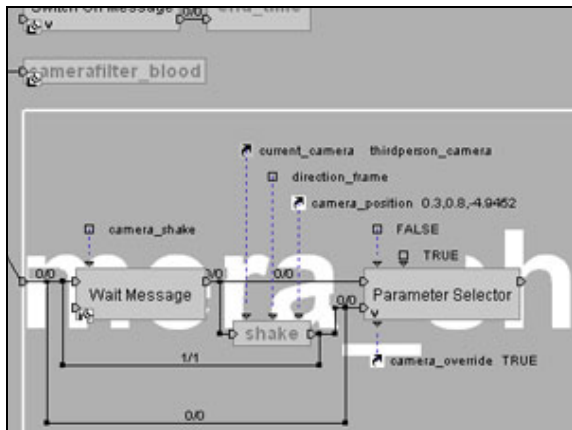
- Create a copy of this last BG and call it "ammo". Specify the right attribute (character_ammo). Replace the Display Progression Bar BB by a Bitmap Text Display BB. Be sure to specify the font in the settings in this BB. Insert the attribute value as the String pIn. To calculate the position of the text (you need a vector2D for this one), create new vector2D parameter (with an initial value of (8, 390) and use a Set Y paramOp and a Subtraction paramOp to change the y value to y_res minus 90. Alternatively, you could use a Set Component BB (you have to change the pOut to vector2D). All the looping and deactivation is the same as for the healthbar BG.

- For the time to be displayed (on top of the timer icon) we'll use something similar as the "ammo" BG. Copy it and remove the Has Attribute BB. Use a shortcut to current_time (found in the score Level Script) as the String pIn. The location should be top_center as well. Use paramOps to do this.

- We just need one more BG called "end_time". When you win the game, your endtime must be displayed in

the center off the screen (among other things). Create a Switch On Message BB (“endtime_on” and “endtime_off”) followed by just a Bitmap Text Display BB (no looping this time, specifying the right location using paramOps). Use a shortcut to current_time as the String pIn. We need different messages for this one because we want the interface off at the moment this is displayed!

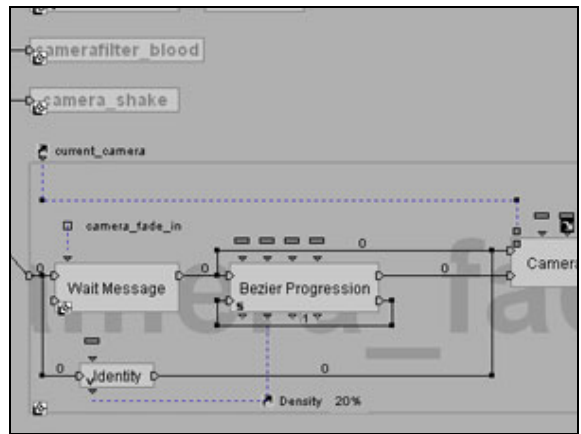
Tip: You can use a Show Mouse Cursor BB to show or hide the mouse cursor during the game. It makes it easier for players to concentrate on the crosshair. However, some players find it very annoying when you hide their cursor when they are playing something in a browser!



19 CAMERA SHAKE When an explosion occurs, we want to have a camera shake that emphasizes this. We already send a message “camera_shake” from the explosion_frame to the Level, now create the corresponding Wait Message BB. From your Behavior Graphs Resources, drag “shake” behind the Wait Message BB and connect it. The BG still needs a couple of parameters in its pIns. The first one needs a shortcut to the current_camera, the second one needs to be a local parameter (or a shortcut) to the direction_frame, and the last one needs to be a shortcut to camera_position (a parameter found in the direction_frame script, the “set camera” BG).

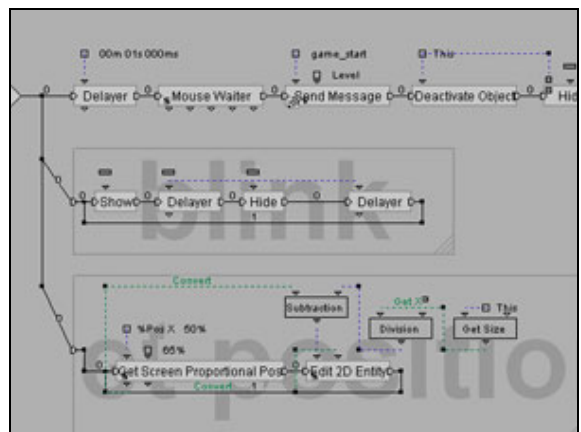
- For this to work, the normal orientation and position calculations that control the camera must be deactivated. Back to the direction_frame script. Create a new local Boolean parameter called “camera_override”. Now we need to insert two Binary Switch BBs to test this parameter. One needs to be the first BB in the orientation_direction_frame BG (connect the True bOut to the Get Mouse Displacement BB, the False bOut can skip the whole thing and must be directly connected to the bOut of the BG! The second Binary Switch must be inserted in the same way into the “set_camera” BG.

- Back in the interface script, we need to make this Boolean parameter False when the “camera_shake” message is received. When the “shake” BG is done, it can be set back to True. Use a Parameter Selector BB to do this. Loop the bOut of the “shake” BG back into the Wait Message BB as well. Now you can create a BG around these BBs called “camera_shake”. Be sure to connect the bIn of this BG to the bIn that sets the camera_override to True as well (so the value is set back to True when the script is first activated).



20 FADE IN One more thing: before our composition begins, we want the screen to fade in from black (for our intro later on). Drag the camera_fade_in BG from the Resources into your script. Create a shortcut to the current_camera as the pIn. This BG uses a Camera Color Filter BB that is activated when the script is activated (so the screen starts out black). If you want to see the rest of your scene, you might want to add a Send Message BB (Message: “camera_fade_in”, Destination: Level) to the “temp” Level Script. However, since we’re going to work on the intro next, this is not directly necessary!

Ok, that wraps up our game interface. It may seem illogical because we’re not quite finished with our game ending, but we’ll first set up the intro to our game.



21 INTRO ELEMENTS Before our game can begin, we first need some sort of introduction! First, the Virtools logo that fades in and out. Then we show our company logo, completely animated (using the Movie Player BB) with sound effect. After that we fade into our 3D Scene and use a pre-animated camera to bring us into the scene. Then we’ll show our game logo, accompanied by an explosion, a camera shake and some sound effects! We also show some credits and a blinking “press mouse to start” button. Next, we’ll switch cameras to create a looping sequence for the camera.

- We’ll use a different approach to the scripting of the intro. We’ll first import most of the elements used from the Resources. Most of these objects have their own little script that controls their behavior. We “only” have to create a script that calls on them when they’re needed. This is different from the rest of the game where we wanted to stay away from little localized scripts and pretty much used global Level scripts. Especially when you want to re-use objects and behavior at different moments in your composition, a localized script is a good way to begin!

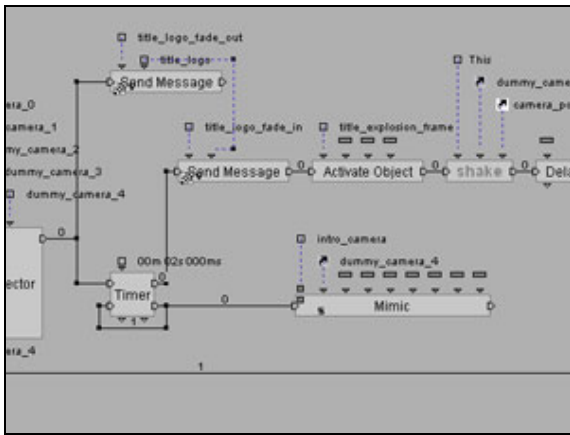
- Drag intro_elements.NMO into your composition. This file contains several 2D Frames (all with the prefix

"title_"), attached materials, textures and scripts. It also contains another particle system (a 3D Frame called title_explosion_frame), some cameras (with animation!) and two sound effects. Give each of the scripts a quick look and you will notice that most of them just make sure the attached object is positioned correctly and is shown when the object is activated (you may have noticed that they are hidden and not Activated at scene start). Some minor things need to be tweaked:

- For the title_end Script you need to specify the crosshair.

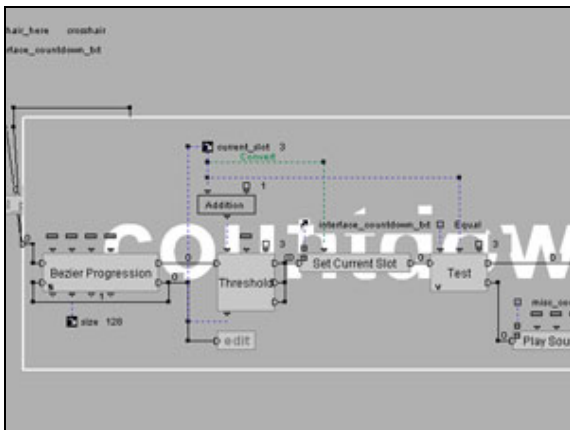
For the title_start Script, make sure that there is a Send Message BB with (Message: game_start) and set the Destination to Level. This will act as a kind of "starting button" for our game!

For the title_explosion_frame (a 3D Frame!) Script you need to specify the explosion texture for the particle system. You also need to specify the two misc_explosion sounds in the lower part of the script. These are already present in your scene.



22 MIMIC CAMERA So, the game will begin using the animated_camera and then we will switch to the intro_camera (first setting the intro_camera on the position and orientation of the animated_camera). This camera has a script that creates a nice looping sequence. Every now and then it picks a different "dummy" camera to mimic. This system uses a Sequencer BB. The movement of the camera is handled by the Timer BB and the Mimic BB.

- For the "shake" BG in this script to work, we need to specify a shortcut to "camera_position", much like we did before, replacing the existing link.

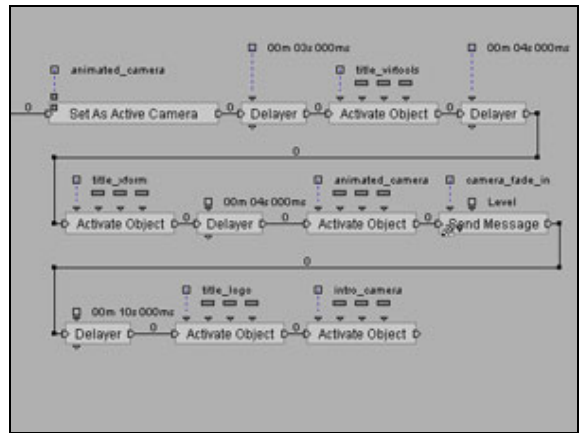


23 COUNTDOWN The only thing missing from the interface now (besides the fact that it doesn't work yet) is a nice countdown before the game begins.

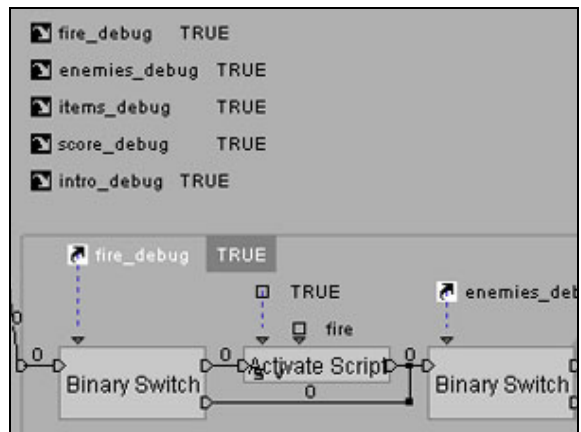
- Drag interface_countdown.NMO into your composition. This is a 2D Frame with a script, an animated texture and two sound effects. Take a look at the script. It uses the Set Current Slot BB to change the animation frame of the texture, every time that Bezier Progression BB Out is activated. This BB was used to change the size of the 2D Frame from 64 to 128. The small BG underneath the Threshold BB makes sure that the 2D Frame appears to be scaling from the center outwards. Without it, the upper left corner of the frame appears to be fixed! The countdown will begin when the object is activated, when the countdown is done it will broadcast a countdown_done message into your scene and it will deactivate itself.



Tip: It's a good idea to use the Broadcast Message BB instead of the Send Message BB when several different objects are waiting for the same message. In the Broadcast Message BB, no specific Destination has to be set.



24 INTRO SCRIPT Now we're going to create a small script that manages the intro sequence. It's just a long sequence of BBs, interrupted by Delayer BBs. First we need to set the animated_camera as the active camera (this doesn't activate the animation yet!). We wait 3 seconds. Then we activate the title_virtools 2D Frame (be sure not to accidentally specify the title_virtools texture!). Edit the BB so that all checkboxes are checked. Wait 4 seconds. Activate the title_xform 2D Frame, again check all checkboxes. Wait another 4 seconds. Activate the animated_camera (all checkboxes). Send the "camera_fade_in" message to the Level. Wait 10 seconds for the animated camera to finish its animation. Activate the title_logo 2D Frame (all checkboxes), followed by the intro_camera (all checkboxes). That's it. Give your scripting a test run (try to disregard the rest of the game!).



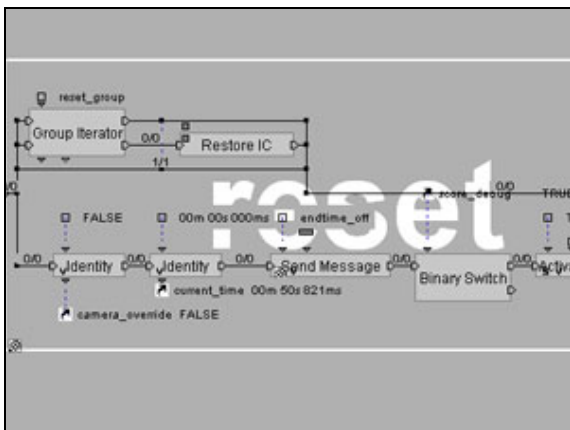
25 MAIN SCRIPT Most games consist of three main parts: the start, the game and the end. We still need a script that handles the main flow of the game

and picks up all of those “game” messages we are sending into the Level. When our composition starts, we first have to create all those dynamic objects (bullets, enemies, items etc.). Then we can play our intro until the game_start message is received. Then we can activate all scripts handling the character and its controls. We can then wait for the game to end. During our end, if the player doesn't choose to play again, the composition must be reset (so the introduction plays again). We're also going to use this script to handle other minor systems (music, sky, lensflare ...)

- Rename your “temp” script into “main. First let's move that annoying Mouse Waiter BB to the “control” script, inside the movement BG on the dummy_character. Cut and paste the “left_button” Boolean parameter there as well. Connect its bIn and bOut to the Switch On Message BB there. Since we're only using the Left Button pOut, you can uncheck the rest in the BBs settings (except Stay Active ofcourse!).

- Back in the “main” script, now disconnect all other BBs that are still there and move them to a lower part of the script (you might be able to use them again later on). Now we're going to create a small BG “start scripts”) that activates all important Level Scripts. First, we need to make sure that all Level Scripts, except “interface” and “main” are not Activated at scene start. Then, back in the “main” Script, create a Boolean parameter for each script (for example: “fire_debug” and “enemies_debug”). We're calling them debug because using the Booleans it will be very easy for us to disable certain scripts for debugging purposes. Now we'll create a whole sequence of Binary Switch BBs and Activate Script BBs so that if a Boolean parameter is set to false, it will skip activating the corresponding script. One little thing: if the “intro_debug” parameter is false, we need to send a “camera_fade_in” message (Dest:Level) to the interface_script (otherwise we wouldn't be able to see anything!).

Note: in the “score” Level Script, inside the “timer” BG, it's not necessary anymore that the Chrono BB is activated automatically. You can delete the link from the BGs bIn to the Chrono BBs bIn.

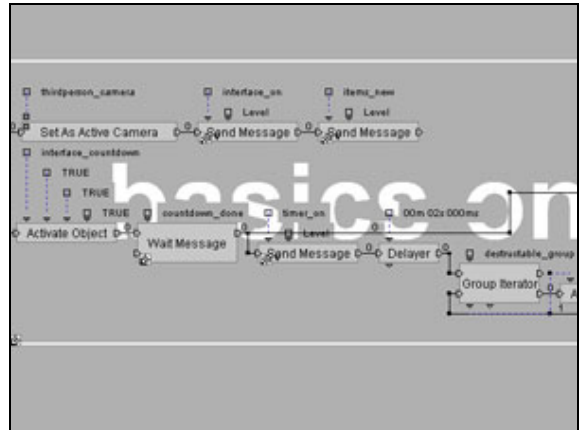


26 RESET BG Underneath all this, insert a Switch On Message BB and connect it to Start. Specify all game messages (“game_start”, “game_over”, “game_outoftime” and “game_congratulations”).

- Now, the first thing that needs to be done when the “game_start” message is received is that we need to reset the ICs on all objects that need to be reset. We'll do this by creating a new group, “reset_group”, that contains all these elements. This group must contain: all destructable vehicles, all spawner vehicles (the vans), the direction_frame, TheHammerAK character, the dummy_character, the thirdperson_camera, the destructable_group (a group can contain groups!) and

the shootable_group. Use a Group Iterator BB and a Restore IC BB to reset all elements. If you haven't done so already, make sure that the Vans start out deactivated.

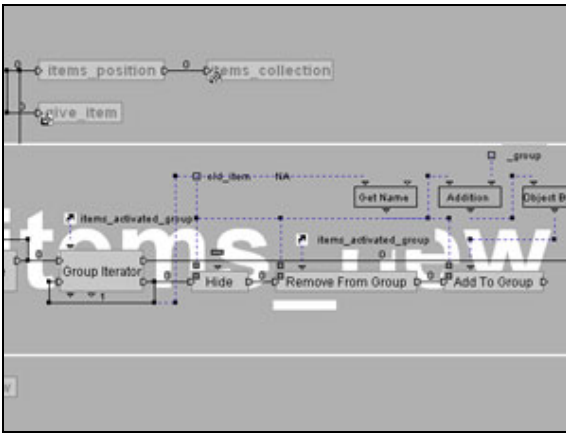
- This BG also gives us opportunity to reset two parameters (camera_override and current_time). We also need to make very sure that the endtime is off (in case we're playing the game for the second time) and that the score script is activated and reset (if the score_debug parameter is True of course!). Create a BG around it called “reset”. You can connect the “start scripts” BGs bOut to the bIn of this new “reset” BG.



27 ACTIVATING THE BASICS Behind the “reset” BG, we need to create another BG that deactivated all things that have to do with the intro: Deactivate the intro_camera, hide and deactivate the title_logo, hide the title_credits, hide and deactivate the title_explosion_frame and hide and deactivate title_start. Call this BG “deactivate intro”.

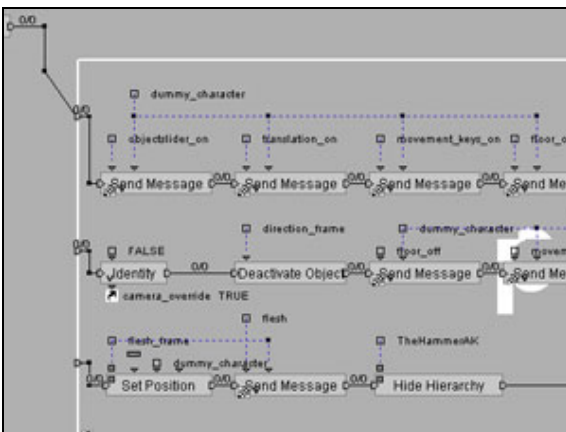
- Behind that, create another BG called “basics on”. In this script we need to set the thirdperson_camera as the active camera. We have to show the interface (by sending a interface_on message to the Level). We also have to send a new message to the Level, called “items_new” (we'll create a BG in the items Level Script that makes sure that items are reset when the game is restarted later on...).

- Create an Activate Object BB (for the interface_countdown 2D Frame, setting all parameters to true). Behind that, add a Wait Message BB (the interface_countdown Script will send a “countdown_done” message). When this message is received we need to send a message “timer_on” to the Level. When this is done, we need to Delay for 2 seconds and then we need to Activate all objects in the destructable_group (using a Group Iterator BB). Be sure to set all parameters to True again. Make sure that at least one BB is connected to the BGs bOut!



28 ITEMS NEED RESETTING In the last step we send a "items_new" message to the Level. We are going to use that message in the "items" Level Script. Create a looped Wait Message BB ("items_new"), and behind that we must create a script that is similar to the "items_collection" BG. Use a Group Iterator BB to put all the items that may be in the items_activated_group back in the group they came from. You can connect this BGs bIn to Start and its bOut to the "items_position" BG (so that new items are put on the locations specified in the array).

- While we're on the subject of items, you might want to check the health you gain when you pick up a health item (in the score Level Script). When full health is 1000, a health item must give you at least 250 health, not 25!



29 PLAYER ACTIVATION When we are ready to begin playing, we need to activate all functionality attached to our dummy object. When the game is over, if the player is out of time or if you've finished the game, the character functionality needs to be disabled. Also, when the game is over or when the player is out of time, the character must die.

- Back in the "main" script. Create a new BG called "player" and give it three bIns. Now, to activate the character we must first send a few messages to the dummy_character (attach these BBs to the top bIn): objectslider_on, translation_on, movement_keys_on and floor_on. Then, behind that, we need to Activate the direction_frame (setting every parameter to True again). Behind that we can add a Identity BB to set the camera_override to "true" (true meaning: you can move the frame with your mouse!).


- For the second bIn of the BG, we need have the same BBs, but in the opposite order (using the "off" messages, setting the camera_override to false and using a Deactivate Object BB instead of the Activate Object BB of course!). You also need to remove The Hammer bodypart from the shootable_group and then,

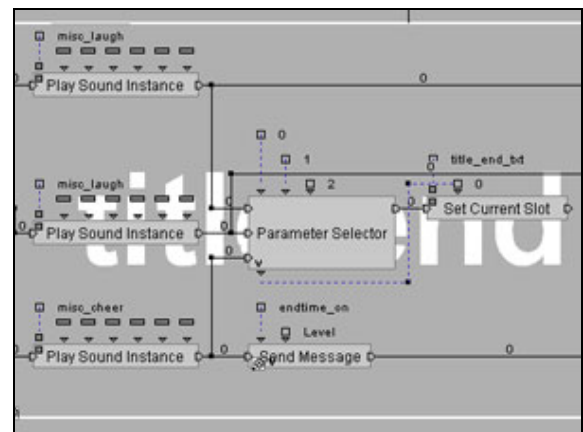
after a one frame link delay, send a "control_reset" message to the dummy_character (so no weird animation or shoot messages are sent after you're dead!).

- For the third bIn (the one that is activated when the player needs to explode) we have to: Use the Set Position BB to set the flesh_frame on the dummy_character, we have to send the "flesh" message to this frame and the last thing we need to do is to use the Hide Hierarchy BB on TheHammerAK character.

- Create one bOut for this BG, connecting the last of the middle row of BBs to it (because this row will be activated for all endings).

- Check if it works. Now it may be a good idea to set a few new initial conditions. We need to set a good starting point for our game. While playing, move your character and the camera to a good starting position (for our game we've chosen a location on which the character is visible at the end of the camera animation (37.3, 0.9250, -4.7). Stop the composition. Now set ICs on the following objects: TheHammerAK (the whole character hierarchy), the dummy_character (the whole character hierarchy), the direction_frame (make sure it's not Activated At Scene Start) and the thirdperson_camera.

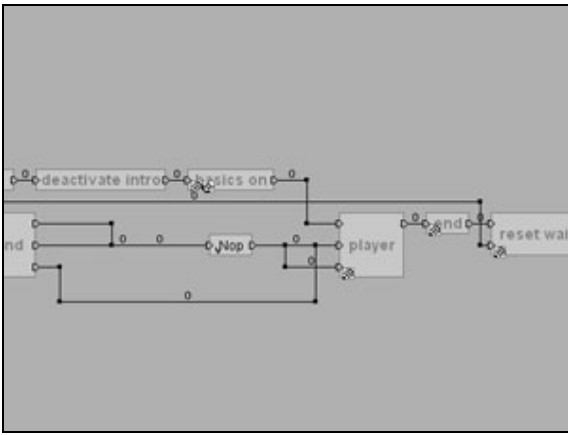
 **Tip:** Use the new debug parameters to deactivate stuff you're not currently testing.



30 CHANGING THE END TITLE The game can end in different ways, that's why the title_end texture has more than one slot. We also want a different sound for different endings. Import misc_laugh.MP3 (a creepy laughter for when the player fails) and misc_cheer.MP3 (for when you win) into your composition.

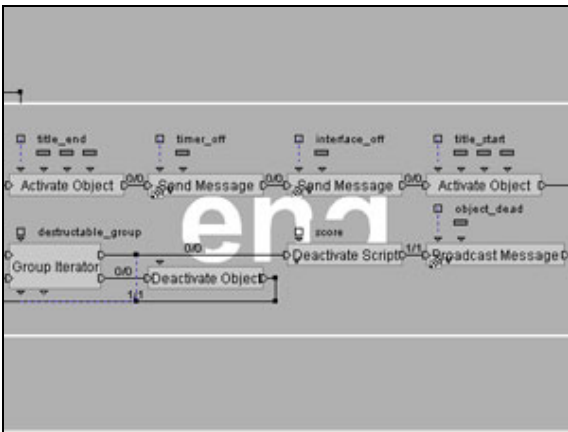
- Create a new BG called "title end" with three bIns and three bOuts. Inside this BG create a Parameter Selector BB (set the type to integer and the values to 0, 1 and 2 (so you have to add one parameter)). Add a Set Current Slot BB (connected to the Parameter Selector BB of course). Add two Play Sound Instance BBs (for the laughter sound, one connected to the top bIn of the BG and one for the second) and add another Play Sound Instance BB for the third bIn of the BG. Make sure that 2D is checked for the BBs and that in the setup of the sounds Streamed is unchecked!. Behind this last BB we can add a "endtime_on" Send Message BB. Connect the BBs so that each bIn of the BG leads to a corresponding bOut.

Note: in the "interface" Level Script, it's best to adjust the position compensation of the end time to 124.8. Otherwise it appears to be a bit off-center.



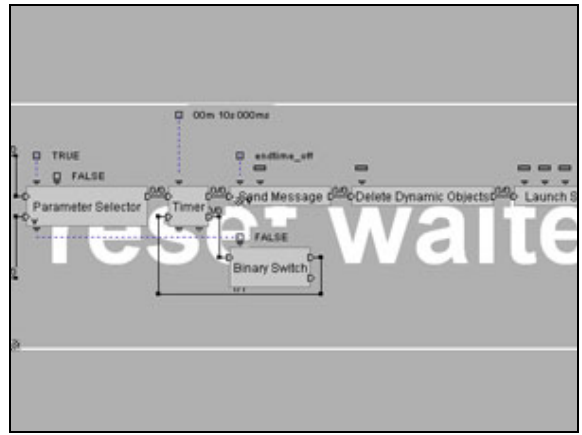
31 ENDING CONTINUED Now we need to connect the bOuts of the title_end BG to the player BG. The top bOuts (for game_over and game_outoftime) both need to be connected to the second and third bIn of the player BG (we've used a Nop BB to "tie" the links together to make it more clear). The lower bIn (for the game_congratulations part of the script) needs to be connect only to the second bIn (no need to make the player explode when you've won the game).

- As you can see in the screenshot, behind the player BG there are already two new BGs, "end" (displaying the (re) start button and deactivating and resetting some stuff) and "reset waiter", a BG that automatically resets the whole composition when the player doesn't press the restart button in time. You can already create and connect these empty BGs. Notice that a lower bIn of the "reset waiter" BG is activated. You need to connect this to the "game_start" bOut of the Switch On Message BB (so the BG won't accidentally reset the composition while you've already started a new game).



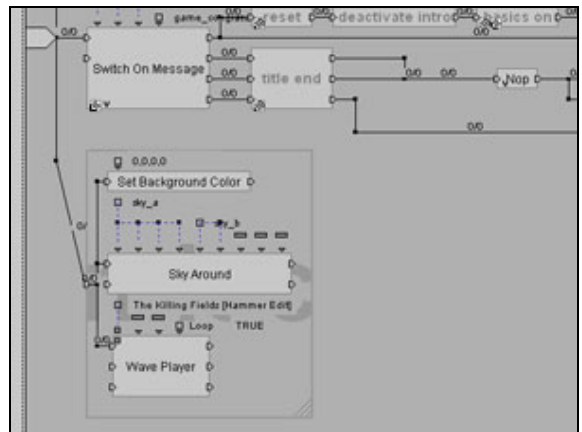
32 END BG Title_end needs to be activated (so that we can see that texture we've just set), the timer must be off, the interface must be off (to give the title_end more impact!) and the title_start frame must be activated (again, all parameters for the Activate Object BB must be set to true). Remember: when this frame is activated, we can use restart the game by pressing the mouse (it sends the "game_start" message). Connect this thing to the bOut of the BG.

- We also need to deactivate all objects in the destructable_group (so that our vans won't spawn any more enemies). After that we can deactivate our score script (we don't want it to send any more game messages!). Then we broadcast an "object_dead" message so that all remaining enemies are killed. Connect this to the Deactivate Script BB with a one frame link delay (making sure that the enemies aren't counted as frags...).



33 RESET WAITER This BG begins with a Parameter Selector BB connected to a Binary Switch BB. This Binary Switch BB is included into a looped Timer BB. This way when the set time is reached, the bOut of the Timer BB is activated, but when the condition of the Binary Switch BB turns false, the loop is broken and nothing is activated.

- When the set time (for example 10 seconds) is reached, the endtime must be set to off, all dynamic objects must be deleted (bullets, items, enemies etc) and the composition must be reset. To do this, we use a Launch Scene BB. Edit the parameters of this BB, setting the Reset Options to "Force Reset". We can leave the rest undefined. Give your composition a good testing!



34 MISCELLANEOUS THINGS We still need to add a couple of things to our composition to make it more colorful. Import The Killing Fields [Hammer Edit].MP3 (uncheck Streamed in the setup) and two textures, sky_a.BMP and sky_b.BMP. Now, in the main script, create "misc" BG. Add a Set Background Color BB (setting the color the black), a Sky Around BB (defining the sky_b texture for the down and up texture and the sky_a for the rest). Also add a Wave Player BB. Specify the soundtrack, setting Loop to True.

- Every game needs a good lens flare. Import flare.NMO into your scene. This file contains an array (specifying some data) and a flare texture. Create a script on the sunlight light. Add a Lens Flare BB, specifying the flare_array and the flare_txt. Set the size to (2,2). You might want to set a new position for the light (so that the shadows in the environment look right...), for example (100, 350, 100).

- If you run your composition now, you have a thumping soundtrack to accompany the action on this sunny day!



Tip: It may be a good idea to set the Far Clipping for your cameras a bit further away (let's say 500 meters). Setting it closer won't

increase performance much in such a small environment.



35 PUBLISHING YOUR GAME After you've finished testing your game extensively, it's time to export to the .VMO format! You can either use the File > Create Web Page option or you can choose File > Export to Virtuol Player. This last option only creates a .VMO which you must embed in your HTML manually. You can take a look at the .HTML file included in the Resources ZIP file for an example.

That's it, you're done! If everything went well you now have a finished 3D game that looks a lot like The Hammer: 2 Minutes Of Mayhem. If there are still some things that didn't go well, don't be let down: you can't get everything right the first time. We hope these tutorials will inspire you to keep creating webgames with Virtuol Dev, or at least gave you some good ideas for future game development. Good luck!

End of part 5.

