How to export units from 3dsmax to Battlezone

By systeme

Two ways exist to export units from 3dsmax to Battlezone, one consists in exporting geos separately using the .geo exporter and the other is a more complex yet more precise way involving the exportation of a group of polygons of interest into a readable format for makeobj.exe then into a single vdf.

Part 1: common steps:

Step 1: Having the right plug-ins:

Before we start, make sure to have the plug-ins matching your 3dsmax version. XSI exporters are available for many versions ranging from 3dsmax7 to 3dsmax2012, while vdf exporters exist for versions going from 3dsmax 4 to 2009 (including max 7 and 9). These latter exporters are directly available on this site.

Once your exporters are put into the plugins folder of your 3dsmax (available in the core location in which you installed your 3dsmax), you are ready to convert your model into a real Battlezone unit.

The first and easiest way to export is by using the geo exporter and includes only this second following step:

Step 2: Exporting the geo:

To export your model this way, you may want to position your geos in 0 0 0 in order to have them spawn there in the vdf loader, otherwise their current position will become the position 0 0 0 and it might be harder to adjust.

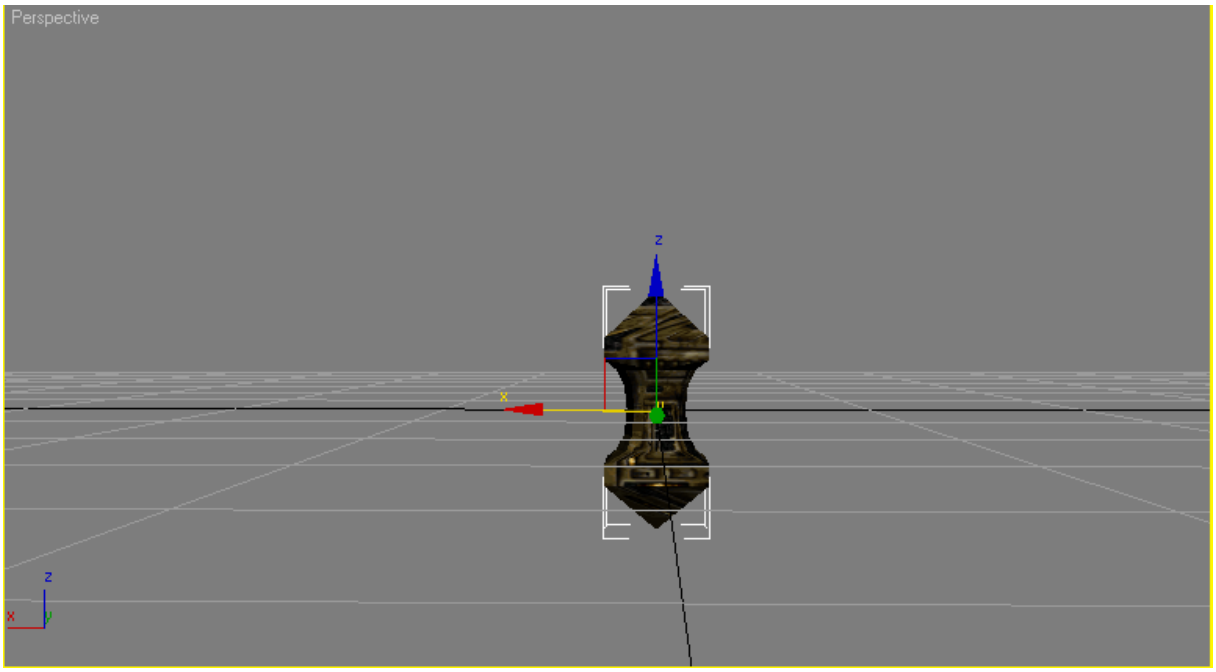


Figure 1: a geo placed in the desired position before exportation.

Once done, you can export your geo directly by selecting the geo format and giving it a name. Make sure your geo stays within the 8 characters limit or it would be impossible to index within your vdf. Last but not least, you have to select a texture that matches the bz texture name style (for example avtank00), or the geo simply won't export.

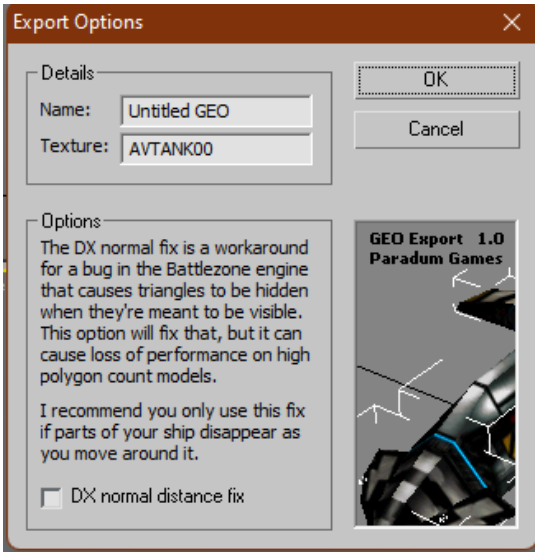


Figure 2: Export Options.

Part 2: the way of Makeobj:

Because of its nature as a developer tool, makeobj might look a bit too primitive and abstract to handle, leading it to be less appreciated than the regular geo exporting. Nevertheless, this .exe let us edit otherwise inaccessible properties on both vdf and sdf models like mass, animation, and geo type. This program can also repair polygons or lower their amount for the model to be ran more efficiently by Battlezone.

Step 1: Verifying controllers:

In order to export to export an animated model, it is mandatory to make sure all controllers are TCB. To this, each element of the model must be verified separately. As such, it might be more efficient to assign the controllers to be TCB by default before starting to make the model. By selecting an object and going to the section visible in figure 3, which is normally at the right side of the screen, this can be easily done. Make sure that all three controllers (position, rotation, and scale) are set to TCB. Once set, we are prepared for the next step.

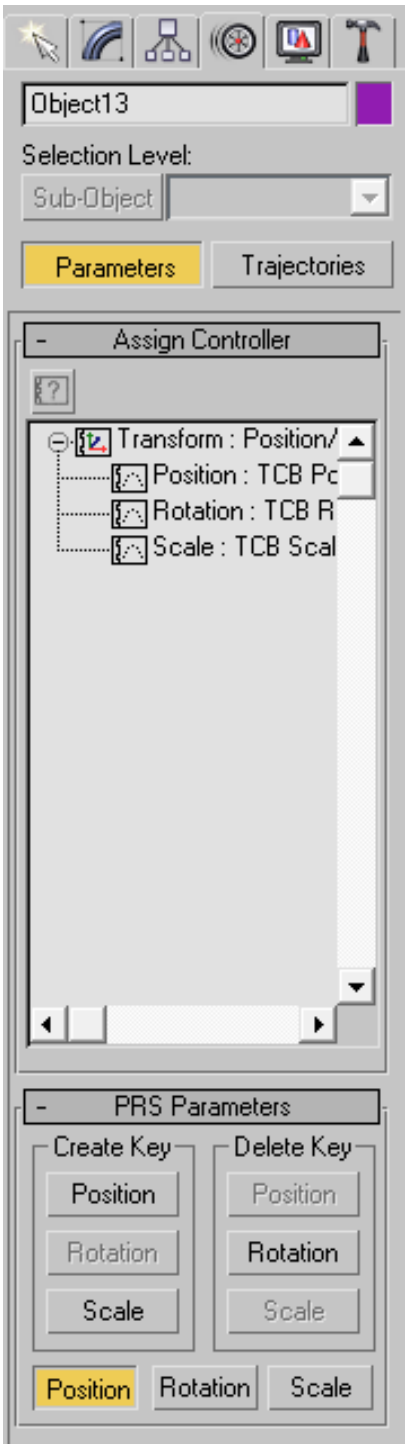


Figure 3: TCB controllers

Step 2: Selecting a Texture:

Because Battlezone only handles textures that are either 16, 24 and 32 bits, it is recommended to have one matching these properties. Additionally, the texture name must not exceed 8 characters to be correctly applied. But, as a way to make it easier to differentiate between each texture, every single of them can be called with a maximum of 6 characters and the last two characters could be the

corresponding digit. Thus, by taking the texture in figure 1 as an example, avtank00 would be the main texture and avtank01 would be the cockpit texture. If the original author were to add wings texture, thrusters and so on, they would be avtank02, avtank03 etc. Furthermore, the 6 first characters should be sufficient to fully describe the entity (here, av are the 2 characters matching the nation and unit type, American vehicle, and tank would be the shape of the model. American vehicle turret would be avturr, American building solar power would be abspow etc.). Another tool could come in handy in order to have ideal textures for battlezone, makemap. Like his object counterpart, makemap is a developer tool and some of its options might be cryptic at times, but it remains a polyvalent and extremely useful program. Because Battlezone uses .map textures and 3dsmax uses .bmp textures, you need to make sure to have both formats available on your computer. To do so, you need to open makemap.exe using command prompt and access the folder in which your bmp exists. Make sure makemap is in the same folder to guarantee the success of the operation. To display the available options, you can simply type makemap.exe in the command prompt and they would be shown there. This program can export in 16-, and 32-bit bmp/map formats by using -565 or -8888 options. While it uses other formats, these are the most commonly used for Battlezone editing. To convert a bmp to map, you can type -565, followed by your texture name bmp. For the other way around, you need to do the following : -bmp, - format, your map name. makemap can also convert an entire folder by replacing the texture name by that of the folder. Make sure to have makemap next to your folder in that case.

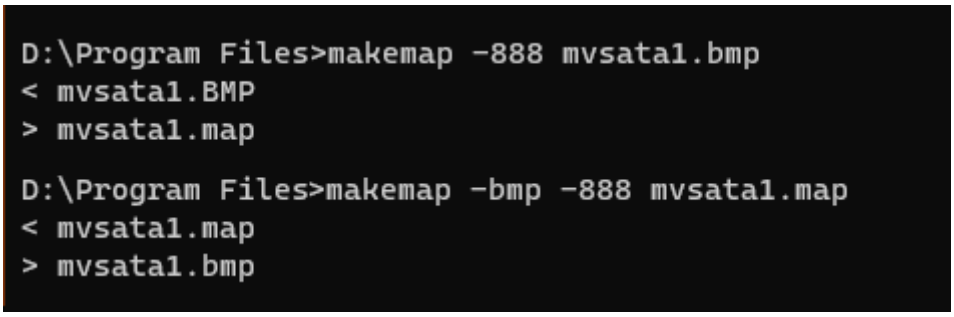


Figure 4: converting a .bmp file to .map and back.

Step 3: inserting the texture in 3dsmax:

This step is rather easy and requires a minimum experience of 3dsmax usage. To pick a texture, simply press “m” which should open the following menu (figure 5)

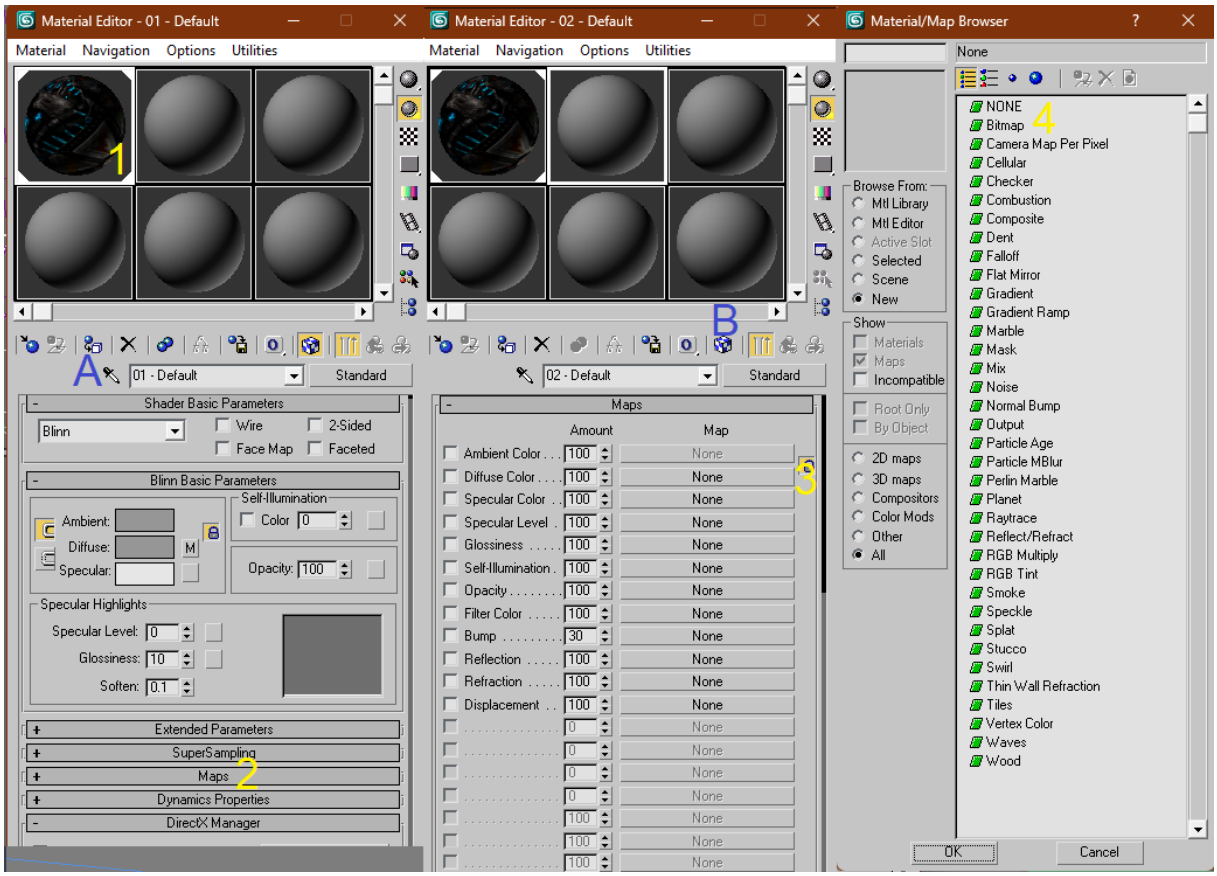


Figure 5: how to use the material editor.

In this menu, you have to select a layer(1), then reach the maps section. Once done, you have to click on maps(2) which would display a large panel of “None” buttons. Select one of the none(3) then choose bitmap in the map browser(4). All what is left to do is to enter your map directory then pick it. When this is done, apply your material to the desired element by selecting your object then pressing “assign material to selection”(A). To effectively display the texture, press “show map in viewport”(B). This latter button can be useful to both enable and disable the texture in case we are using a complex bitmap that wouldn’t allow us to see certain polygons/edges/vertexes in the 3dsmax view.

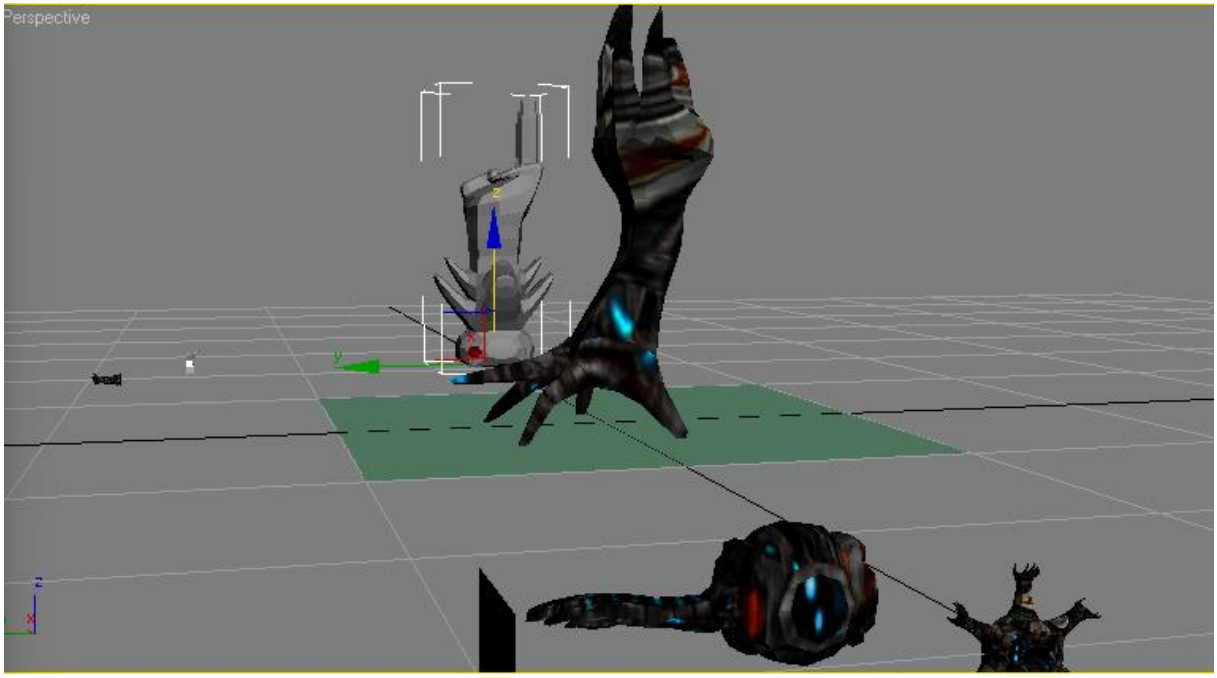


Figure 6: Several textured objects, the selected object has its texture temporarily disabled.

Step 4: Making sure the parameters are up to date:

Because sometimes the transformation parameters fail to be refreshed, it is important to reset them manually in order to avoid misfitting elements in the model. In some cases, parts might be way too big, too small, or too far. To fix this, proceed to click on the hierarchy tab in the menu on the right, then in the adjust transform section, press the “transform” and “scale” buttons. This operation might require several clicks and each button pressed will count as an action. Hence, it is highly recommended to have several saves as you might not be able to go back to a previous state after resetting the transforms.

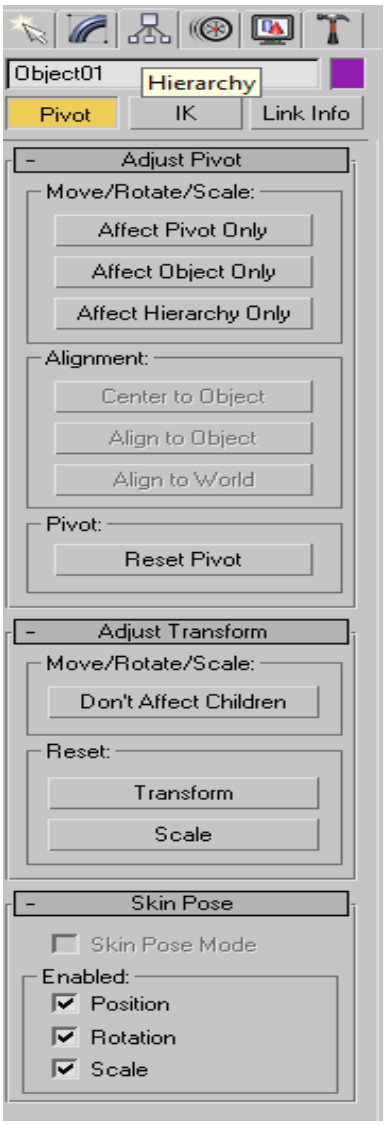


Figure 7: Hierarchy

Step 5: Adjusting the pivots:

In the same menu, it is possible to change the pivot of your object. As its name suggests, the pivot is the base point affecting the rotation of the object and all elements attached to it. The pivot could be put anywhere (not necessarily within the object itself) and will also be used by Battlezone to determine its movement once the unit explodes. Furthermore, the pivot's orientation can also be changed, and determines the in-game rotation of the element or other functions such as smoke emitting for thrusters or the angle of view for the pov once the player is inside the unit.

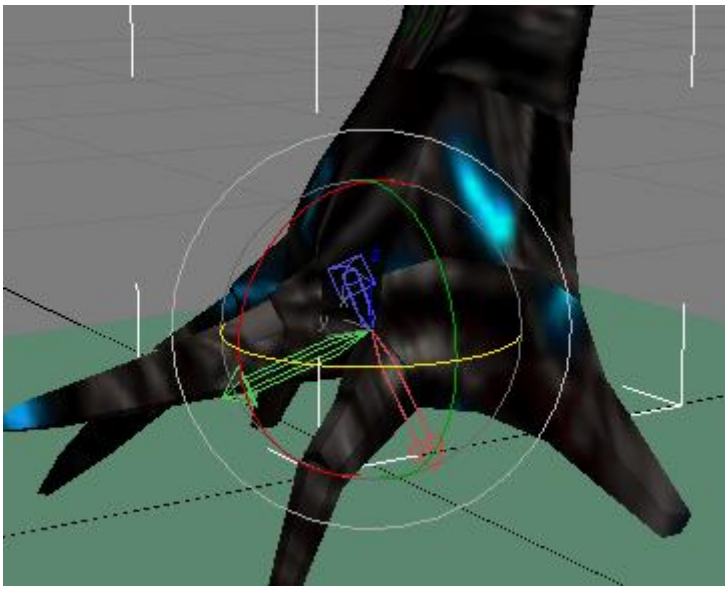


Figure 8: the x y z axes of the pivot

It is best to go through these steps before setting the actual hierarchy of your model.

Step 6: Geo naming and parenting:

Among the most important parameters in Battlezone models are the geo names and their parents. Incorrect naming would lead to the geo not being read by the game and an incomplete model in the best case. As well, incorrect parenting would lead to the geo not behaving correctly (for example a hardpoint geo won't move along with the turret) and error messages displayed every time the model appears in game, be it by normal spawning or exiting a vehicle. Luckily, a window exists in 3dsmax that shows the complete parenting tree of your model as well as the name of each element for easier tracking: the schematic view (figure 9). In this mode, we can easily link, unlink, and rename geos all while keeping track on them and without risking moving them inadvertently. To rename the geo, simply double click on its current name and its frame will change into a text bar. It is also possible to rename it in the menu on the left of your 3dsmax.

Linking and unlinking geos can also be done in the regular window on the upper left section. Two buttons are available, select and link and unlink selected. To link, you have to click on the desired geo then drag the cursor to its parent. This can also be applied to groups. To unlink a group or a single geo, simply select it and click on the unlink selection. Those two buttons have equivalents in the schematic view (connect and unlink selected in the top left corner) and work in a similar way, but only within the schematic view window.

Geo naming follows a certain logic which is limited both by the 8 characters rule and the role of the geo itself. Thus, it is per use that geos are named according to the following: first three letters of your unit, geo level, and geo type (for example asp11bda would be aspilo.vdf geo, level 1 or immediately visible and bda which is the first geo). The most used geo levels are level 1 (referenced as 11) which can be seen from an external point of view, level 2 (referenced as 21) which can be seen from the internal point of view and are generally parts of the cockpit, and level 3 (referenced as 31), that are untextured and low poly elements for the internal point of view. If your geo is a regular "body part" its name could end with bdN, n being a letter or a number if you have more than 26 geos, although other terminations could be used, such as blN. Other roles can be turret, either txN for up/down axis or tyN for left/right axis, this latter axis isn't really visible in multiplayer due to a glitch that is pending a fix but can still be useful in single player missions (please note that tx/ty and other animated geo types do not require any further animation move/rotate in-game). In addition, some geo types that are strongly recommended to have yet are not visible are pov, which is your unit's point of view (the unit might glitch or crash the game if this geo is not featured), gTX which is the gun part, T being the letter matching the hardpoint (c for cannon, r for rockets, s for special and m for mortar) and X the gun number (gc1, gc2 etc.), smX or smoke point emitters and enX for geyser emission on producers. Please note that the same model can have a huge number of hardpoints even if the in-game unit can only have up to 5 guns. This could enable us to use the same model for different purposes such as a scout that begins with only two cannons then would evolve through the campaign to feature a rocket or a special hardpoint without having the need to make several vdfs. Furthermore, gs1 determines other properties of your unit such as the anchoring point on your tug. When making an internal geo (21) make sure to give it a name that matches its lvl 1 part, to link it directly to the said part, and make certain both pivots are on the same exact spot (even if the object itself is elsewhere) or it won't appear in-game. Last but not least, keep in mind that splinters, flares, weaponmines and such do not require any gun geo, as their base geo (usually bda) determines the point of emission. The first character of your geo can only be a letter. A digit or any other symbol will cause makeobj not to be able to read it properly and might not export your geo properly or even crash.

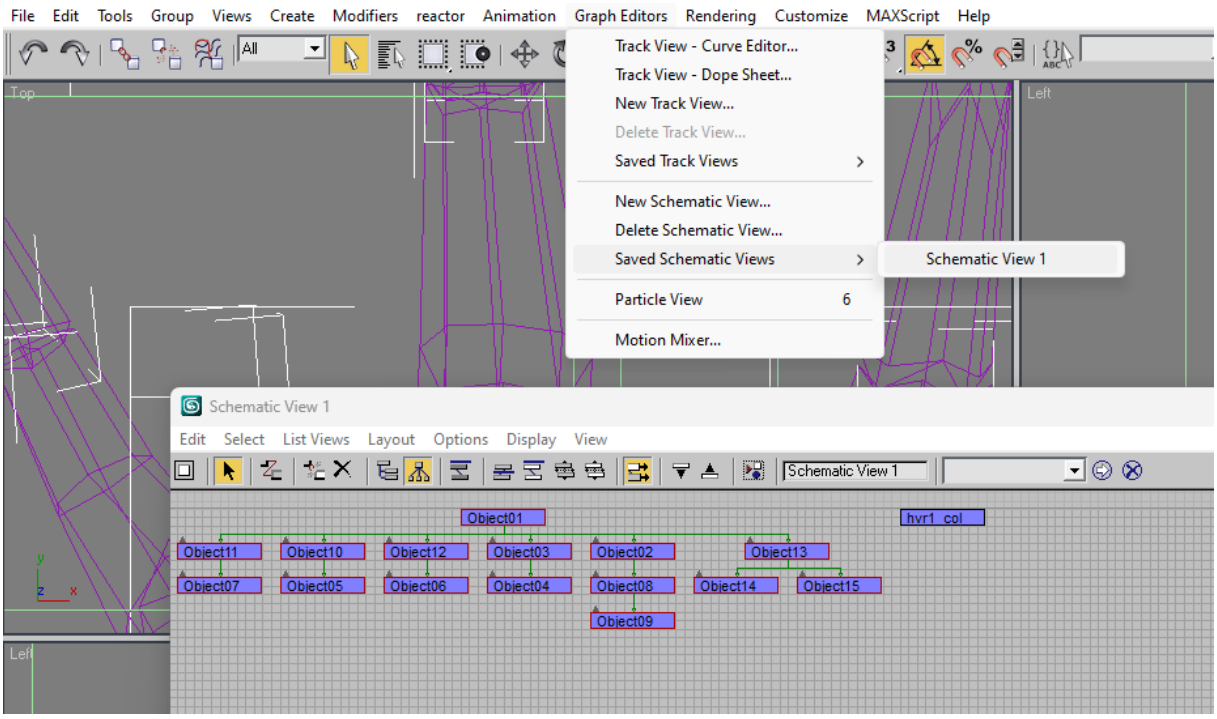


Figure 9: Schematic view and where to find it. We can see that some geos are already arranged within a tree while one isn't.

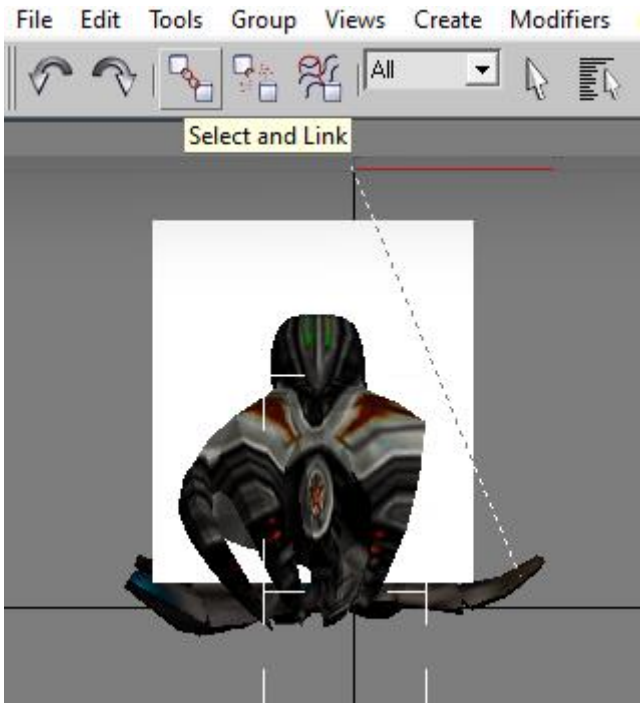


Figure 10: how to link directly in 3dsmax.

Step 7: Collision and dummyroot:

Collision is the trickiest part of the model editing because nor 3dsmax or makeobj can generate it automatically for units. However, for production units, guntowers, and buildings, collision is generated in-game and is geo-based. The best example is the NSDF supply depot which consists in 4 separate elements. While you can clearly go between its geos, each of them has its own boundaries and you cannot get inside them. If this building were to be a unit, however, it would be tedious to get into the space that is between these objects. This is because non-producer vdfs rely on a single collision box which is manifested by a geo type called XXX1_col that does not appear in the final model and is not generated as a geo by makeobj. The only way to reliably obtain this geo is to import a .3ds model that already has it. While it is strongly unadvised to rotate this element to avoid errors in makeobj, we can easily reshape it and move it around to best fit our unit. You may also notice this geo is “lacking” faces, but it is not really the case because each face is strongly distorted and flipped. According to this paragraph, the good news is you only need this collision geo for units that are not producers or guntowers.

Dummyroot, on the other hand, is the origin surface for your model. Because makeobj cannot guess the height at which your model is, it would use the lowest vertex as the origin point. By creating a dummmyroot in 3dsmax, you can make your model hover (which is more useful for buildings) or slightly sink into the ground (in case you want to simulate a heavy unit that will make the ground under it collapse when not active) by positioning your model accordingly, were it to be above or through the dummyroot. As evidenced above, all models might require a dummyroot. This element is always the highest point of your hierarchy, which means the main parent (usually the bda) must be linked to it. If you are to export multiple units at once, make sure all the main parents are linked to the dummyroot.

To make a Dummyroot, simply create a plane and name it so.

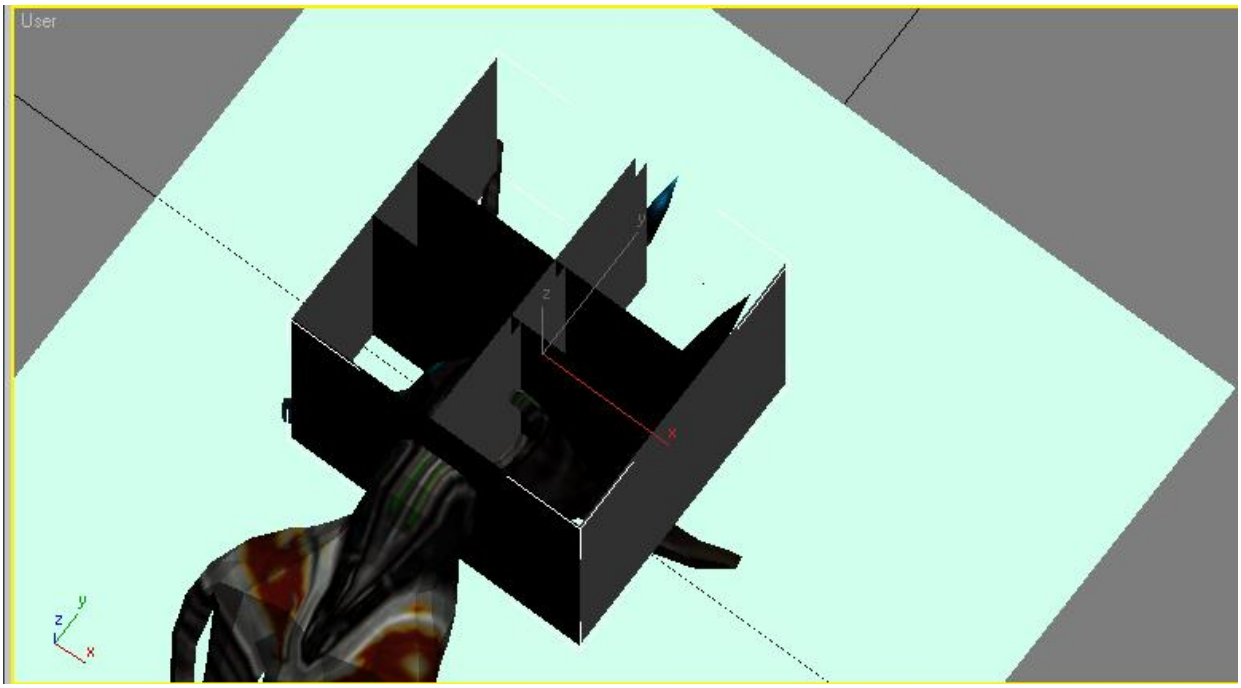


Figure 11: hvr1_col and dummyroot

Step 8: Animation:

If we followed step 1, 4 and 5 correctly, it is time to do the animations. Battlezone only handles two types of animation: rotation and position. Yet, size and twisting can be simulated by hiding parts of a geo into another, giving the illusion that the same part is growing or being distorted. To make sure we only use these two types of animation, a key filters button (Figure 12) can be seen at the bottom right of your 3dsmax main window. All what is left to do is to toggle the two categories and we're good to go.

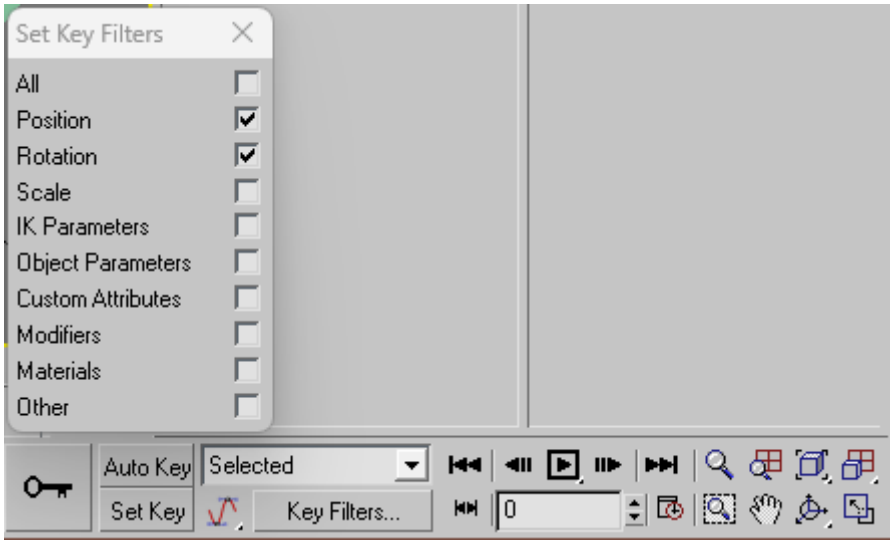


Figure 12: Key filters, key and autokey

As to not inadvertently permanently transform our model, it is highly suggested to use the button AutoKey, located in the same section as Key Filters. By doing so, the bar on the bottom of the 3dsmax window will be red. Henceforth, every operation (within the filters) that is done to a geo will generate a key, determining the animation to be followed from the starting point to this moment. A tip to follow would be selecting all of your geos at the frame 0 and press the key looking button in order to generate our first key, diminishing the risk of losing the original shape of the model in case an origin key fails to be automatically generated. Keys can be selected when clicked on and moved anywhere on the timeline. They can also be duplicated by pressing shift right before starting to move them and several keys can be selected at once by pressing ctrl. Keep in mind that when making a loop animation ("moving forward" for a walker for example), the last key must match the first. Approximatively 25 geos can be animated on the model before glitches start to occur and an ideal frame amount should be between 10 and 20. The same frame can be used for two animations (for example if the unit stands up from frame 1 to 10, it can have an idle animation from frame 10 to 20 without any error occurring). While geos can be seen moving erratically between frames sometimes, keep in mind that this only applies to the model in 3dsmax and not the final vdf, but generating a key on an interesting "erratic" frame would make it permanent (this would be useful in case you want your unit not to stand still while idling for example, but if you do not press the key button, it would remain still in-game). Finally, while the amount of animable geos is limited, elements attached to them would move as well. Thus, when moving the arm of your walker or the cockpit of your turret, the gun hardpoint/internal cockpit would move along, sparing further animation for your other geos.

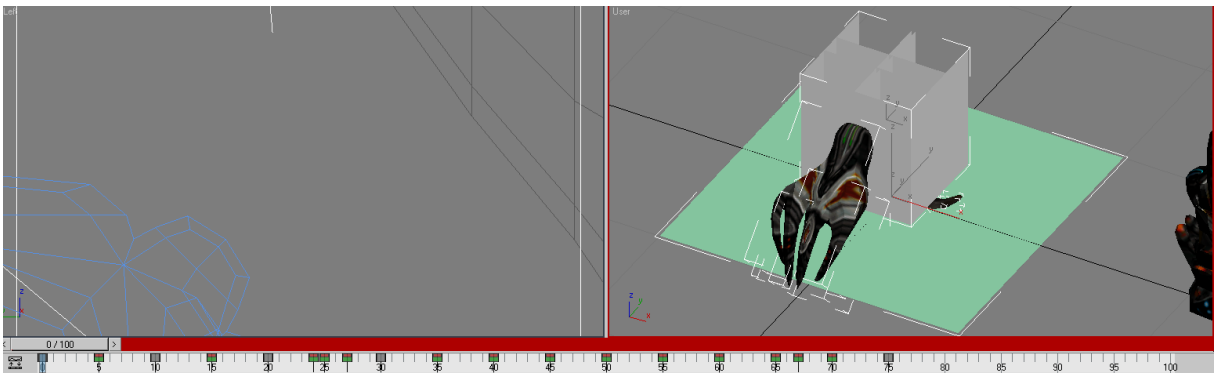


Figure 13: AutoKey. You can see the bar turning red. The "ruler" below allows us to scroll between frames. Here we are at frame 0. Each box on it is a Key.

Figure 14: duplicating the keys.

Step 9: Exporting:

If all is set, we can begin exporting. All that we have to do is to select all the desired geos and to click on “export selected” then save as xsi. This particular format is unfortunately a mandatory step and requires a program called 3dexploration. Make sure to toggle the animation case if your model is an animated model when exporting. For the exportation to work correctly, all geos must be linked together, with the main geo being linked to the dummyroot. All the linked geos must be exported, or errors might occur. if you have two models in your 3dsmax and you only wish to export one of them, you have to unlink all part of your second model from the dummyroot or any other exported element, lest the exported files become corrupted.

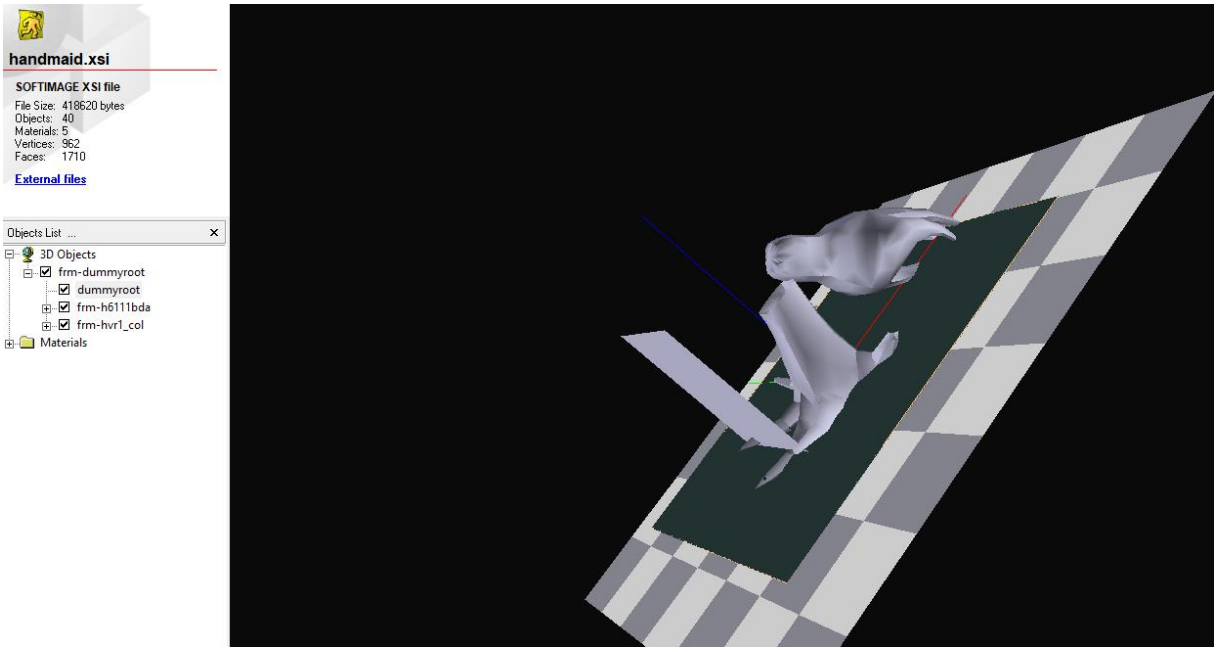


Figure 15: the model has been successfully exported.

After exporting to xsi, we have to turn it into .x using 3dexploration. This format is the only format that can be read by makeobj, even if it states that it can read .3ds (this feature has not yet been implemented). Several parameters can be changed in the .x itself like the texture or animated elements. This format is readable directly and a few parameters are to be edited in notepad in case your model is animated. The current format contains sections called AnimationOptions and does not have the geo name at the end of its regular animation section, yet makeobj needs that one parameter and cannot read AnimationOptions. Thus, it is important to correct these parameters according to figure 17. Luckily, this process has been automated and your x can become makeobj-friendly by using xprep (which can be found on the maproom). All that needs to be done is to choose a

file an submit. It would then automatically download a file called result.x which needs to become that of your model.

Figure 16: xprep.

It is also important to notice that AnimationSet section simply does not exist for non-animated models who do not require them (tx/ty and other animated geo types won't create this section yet would work properly) and this might indicate if your animation has been properly exported as well.

handmaid.x

Fichier Modifier Affichage

```
  0;
  9;
  10; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  15; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  20; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  24; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  25; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  27; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  30; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  55; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  75; 4; 1.000000, 0.000000, 0.000000, 0.000000;;;
   }
  AnimationKey {
  1;
  1;
  0; 3; 1.000000, 1.000000, 1.000000;;;
   }
  AnimationKey {
  2;
  9;
  10; 3; 0.865291, 4.686232, 2.545863;;,
  15; 3; 0.865291, 4.686232, 2.545863;;,
  20; 3; 0.865291, 4.686232, 2.545863;;,
  24; 3; 0.865291, 4.686232, 2.545863;;,
  25; 3; 0.865291, 4.686232, 2.545863;;,
  27; 3; 0.865291, 4.686232, 2.545863;;,
  30; 3; 0.865291, 4.686232, 2.545863;;,
  55; 3; 0.865291, 4.686232, 2.545863;;,
  75; 3; 0.865291, 4.686231, 2.545862;;;
   }
      AnimationOptions {
            0; 0;
      }
  }
 }
```

Ln 1, Col 1

handmaid.x

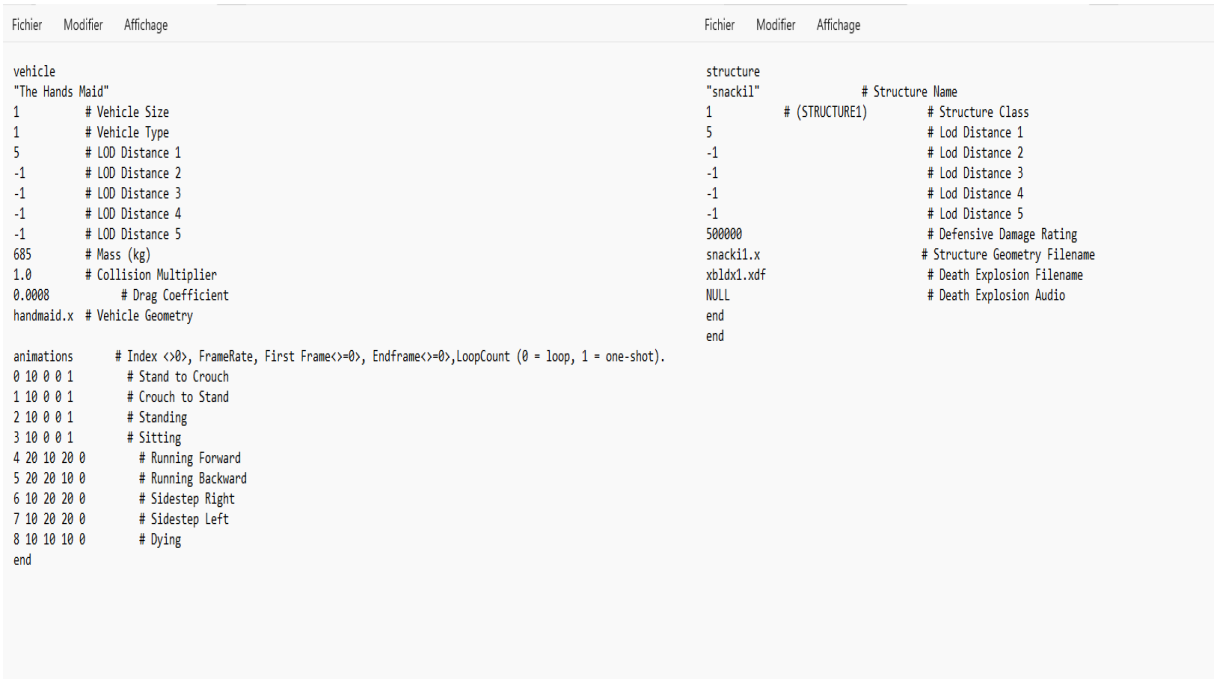
Fichier Modifier Affichage

```
 }
Animation anim-h6111bdm {
  {frm-h6111bdm}
  AnimationKey {
  0;
  9;
  10; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  15; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  20; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  24; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  25; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  27; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  30; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  55; 4; 1.000000, 0.000000, 0.000000, 0.000000;;,
  75; 4; 1.000000, 0.000000, 0.000000, 0.000000;;;
   }
  AnimationKey {
  1;
  1;
  0; 3; 1.000000, 1.000000, 1.000000;;;
   }
  AnimationKey {
  2;
  9;
  10; 3; 0.865291, 4.686232, 2.545863;;,
  15; 3; 0.865291, 4.686232, 2.545863;;,
  20; 3; 0.865291, 4.686232, 2.545863;;,
  24; 3; 0.865291, 4.686232, 2.545863;;,
  25; 3; 0.865291, 4.686232, 2.545863;;,
  27; 3; 0.865291, 4.686232, 2.545863;;,
  30; 3; 0.865291, 4.686232, 2.545863;;,
  55; 3; 0.865291, 4.686232, 2.545863;;,
  75; 3; 0.865291, 4.686231, 2.545862;;;
   }
  }
 }
```

Ln 1, Col 1

Figure 17: animated .x file before and after "fixing".

To make sure your model has all the required parameters to be fully implemented in-game, you need a .txt file with that information. An ideal text file would look like figure 18:



Fichier Modifier Affichage

```
vehicle
"The Hands Maid"
1          # Vehicle Size
1          # Vehicle Type
5          # LOD Distance 1
-1         # LOD Distance 2
-1         # LOD Distance 3
-1         # LOD Distance 4
-1         # LOD Distance 5
685        # Mass (kg)
1.0        # Collision Multiplier
0.0008          # Drag Coefficient
handmaid.x  # Vehicle Geometry

animations      # Index <>0>, FrameRate, First Frame<>=0>, Endframe<>=0>,LoopCount (0 = loop, 1 = one-shot).
0 10 0 0 1        # Stand to Crouch
1 10 0 0 1        # Crouch to Stand
2 10 0 0 1        # Standing
3 10 0 0 1        # Sitting
4 20 10 20 0        # Running Forward
5 20 20 10 0        # Running Backward
6 10 20 20 0        # Sidestep Right
7 10 20 20 0        # Sidestep Left
8 10 10 10 0        # Dying
end
```

Fichier Modifier Affichage

```
structure
"snackil"               # Structure Name
1          # (STRUCTURE1)          # Structure Class
5                                  # Lod Distance 1
-1                                 # Lod Distance 2
-1                                 # Lod Distance 3
-1                                 # Lod Distance 4
-1                                 # Lod Distance 5
500000                             # Defensive Damage Rating
snacki1.x                         # Structure Geometry Filename
xbldx1.xdf                         # Death Explosion Filename
NULL                               # Death Explosion Audio
end
end
```

Figure 18: text file for an animated unit (left) and unanimated building (right)

The header determines what type of object we have (unit/structure) and will therefore determine which model type will be created by makeobj (vdf/sdf). Right below is the unit name for the model, this latter will only be seen in vdfloader/sdfloader and will appear at no point in the game. Lod distances, vehicle size and type are generally the same for every model, so we can leave them as they are. Mass, however, can be changed for units (the default mass is 1750 for units and 200 for persons/powerups). The greater the mass, the stronger collision damage would be. Keep in mind that this also impacts damage received by your unit when falling, which means that impossible mass would inevitably destroy it the second it enters in contact with the ground and also that an incredibly light weight will make it way less affected by land collision, in exchange of what it would be prone to be pushed away or smashed by other units when colliding. The other parameter of interest is the geometry filename, which is the name of the .x we just exported. It can be another name in case you want to make another model with these same parameters. Finally, the animation parameters are, as their name suggest, the parameters that determine the animation type, the amount of frames per second, the starting frame, the ending frame and whether or not the animation is looping (in this order). In the current figure are displayed all the existing animations for a pilot, this unit type being the one with the maximum amount of

animations. Deploying units (turrets, scavengers, howitzers, tugs) only require animation type 0 and 1. Deployed producers and buildings might also need animation 4 (which occurs when they're deployed) and armories use animation 5 when they launch a powerup.

If you are exporting isolated geos, you don't have to make a .txt file. Also, isolated geos can be named with any name that does not exceed 8 characters. This is because makeobj needs the correct naming only to give the geos a purpose within the model you're exporting.

Once everything is set, it is time to finally obtain our game model. For this, we have to open command prompt again and reach the makeobj directory (make sure your x and txt are in the same directory as makeobj) and to ask it to read our unit text file (figure 19). If we are exporting isolated geos, we can ask it to read the .x only (we would use: makeobj handmaid.x) and it won't make a vdf or a sdf.

```
D:\Program Files (x86)\Nouve>makeobj handmaid

3: 6, 7
4: 8, 9
5: 10, 11
Reduced from 12 to 6
. . . . h6111gr1:   24 vertices,    6 faces
Saving h6111bda.GEO
Saving h6111bdb.GEO
Saving h6111bdc.GEO
Saving h6111bdd.GEO
Saving h6111bde.GEO
Saving h6111bdf.GEO
Saving h6111bdg.GEO
Saving h6111bdh.GEO
Saving h6111bdi.GEO
Saving h6111bdj.GEO
Saving h6111bdk.GEO
Saving h6111bdl.GEO
Saving h6111bdm.GEO
Saving h6111bdn.GEO
Saving h6111bdo.GEO
Saving h6111gr1.GEO
Saving h6111pov.GEO
Saving h6111sm1.GEO
Warning...overriding animation 0 in VDF file '0 10 10 0 1'
Warning...overriding animation 1 in VDF file '1 10 0 10 1'

Processing Complete!
```

Figure 19: the command and the final part of the exportation. If everything worked accordingly, it should display process complete!

Step 10: Adjusting the final details:

Additional fixes can be done in vdf loader, such as rearranging the collision or placing the geos correctly. This step is common for the two exporting methods. Collision can be edited first by showing it in view > render style > show collision then edit > collision data. On a side note, in vdf/sdf viewer, y and z are inverted, which means y is used for the height. Furthermore, sdf viewer saves as sdfcp while Battlezone uses sdfcn, this parameter can be changed by using a hexeditor on the file itself.

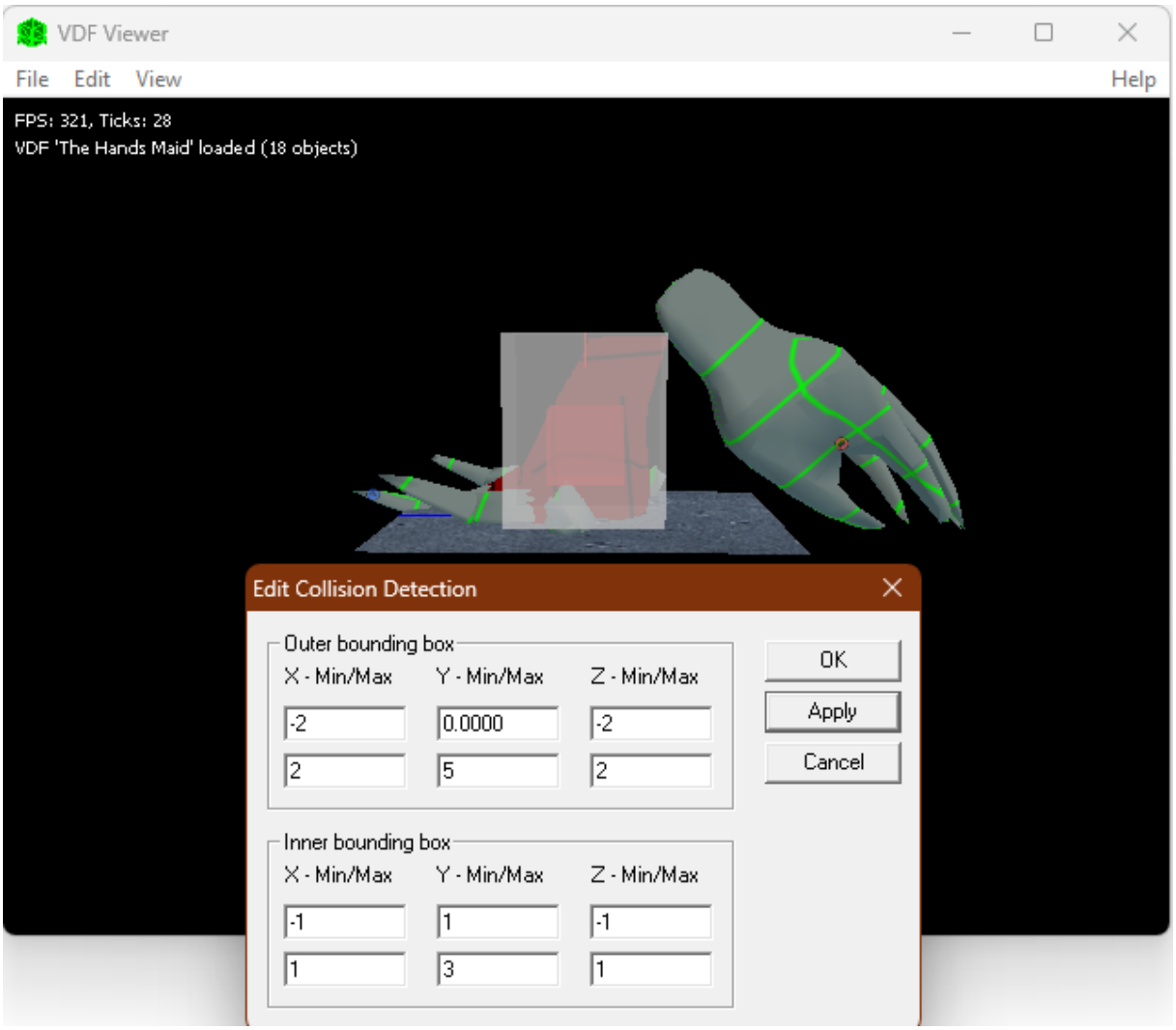


Figure 20: adjusting collision.

On making a sdf model, makeobj will create an additional file called .lin. When it comes to buildings, the geo name cannot determine the type, so the .lin lets makeobj identify them and change their matrix. In sdf, geo types aren't really interesting, but some can still be useful, such as spinners (a geo type that will be permanently rotating). But let's take a look at the .lin in figure 21:

```
Name:    b9511blz
  DDR:   500000
  Class: CLASS_ID_SPINNER
  Flags: 0
  x:     3.000000
  y:     0.000000
  z:     0.000000
  Time:  0.000000

Name:    b9511bly
  DDR:   500000
  Class: CLASS_ID_SPINNER
  Flags: 0
  x:     0.000000
  y:     -0.300000
  z:     0.000000
  Time:  0.000000

Name:    b9511blx
  DDR:   500000
  Class: CLASS_ID_STRUCTURE_GEOMETRY
  Flags: 0
  x:     0.000000
  y:     0.000000
  z:     0.000000
  Time:  0.000000
```

Figure 21: svtelpa.lin file.

Each geo can be identified by its name, followed by its Class. The default geometry class is STRUCTURE_GEOMETRY and needs to be manually replaced by a type of interest (SPINNER in this case). We can then proceed to choose the axis on which it will rotate, either X, Y, Z or a combination of these in case we need the geo to spin on a diagonal. The geo can then spin clockwise (+) or counterclockwise (-). After we're done, we have to reprocess the .x with the new .lin. To this, simply retype the same command line and makeobj will automatically read the new file. If the process is successful, the .lin's data will remain the same as we previously entered them. Otherwise, the x y z parameters will be reset (0.000000) and the edited classes will become CLASS_ID_NONE, requiring us to do the operation anew. To avoid this, it is best to copy and paste an already existing entry and only change the name and the x y z parameters while leaving the DDR and the class untouched. An example of .lin containing spinner class is available in this tutorial.

If you have reached this point, it means you have successfully made a Battlezone model. Congratulations! All what's left to do is to give it an odf and enjoy it within the game.

Important note: While these steps work for both battlezone 1.5.2.27u1 and battlezone98redux, you need to be careful to make the other files that are needed for redux (.mesh, .skel etc.) or your unit won't be visible.

Sources used:

3dexploration, available from https://www.fileplanet.com/archive/p-33401/3D-Exploration-version-1-7-Trial

3d studio max 9

Vdf viewer, available from https://bzmaps.net/misc.php

Makeobj, available from https://bzmaps.net/misc.php

Makemap, available from https://bzmaps.net/misc.php

Xprep, available from https://xprep.pages.dev/

Handsmaid, the unit in this tutorial, available in The Walk.

Svtelpa.lin, available from https://bzmaps.net/misc/tutorials/svtelpa.lin