Malignus’s **How to Map for Homeworld 2** v. 1.4

Make sure you do this in order; the map files are mostly divided into DetermChunk and NonDetermChunk portions, and it matters what goes where.

1. Create the map directory.

Create a series of directories in your Sierra\Homeworld2\Data folder, called LevelData\multiplayer\deathmatch. This is where you should store all custom maps, since this is where HW2 will be looking for them.

2. Create a “.level” file.

Open up Word Pad. Save a new file as “xxx.level” where xxx is whatever you like.
Tip: have Windows always open .level files in Wordpad.
Tip: don’t name it Test.level—HW2 does not like this for some reason.

3. Add In-Game Level Information.

This is stuff that will show up in multiplayer menus in-game. Add in this text:

```
levelDesc = "xxx"
maxPlayers = y
```

Whatever you type instead of xxx will be the **level name** that is shown in-game.
The value you type in for y will be the **number of players** required to play the level.
Tip: While still testing the level, you should set the name to “A” so that it will be selected by default when you start a CPU skirmish (provided it’s a 2-player map, at any rate).

4. Add Player Information.

Add in this text:

```
player = {}

player[0] = {
  id = 0,
  name = "Vaygr",
  resources = 1500,
  raceID = 1,
  startPos = 1,
}

player[1] = {
  id = 1,
  name = "Hiigaran",
  resources = 1500,
  raceID = 0,
  startPos = 1,
}
```

If you want to have more than two players, just add extra players, and remember: The number in player[X] should always be the same as the id = X number. StartPos should always equal 1. RaceID should oscillate between 0 and 1 like you see above. The other values should be mostly left alone, as they are handled by in-game selections.
Tip: to add comments in your .level file, simply precede a line with two hypens: --

5. Add Player Start Point Information.

Now add in this text:

```
function DetermChunk()
  addPoint("StartPos0", {X, Z, Y}, {A, B, C})
  addPoint("StartPos1", {-X, Z, -Y}, {A, B, C})
```

“StartPos0” is the start position for player[0], “StartPos1” is the start position for player[1], and so on up to StartPos5 for a six-player map.

The first brackets, the {X, Z, Y} ones, determine the **location** of each start point **relative to the origin** (the point where the crosshairs meet in sensors manager when the map first starts).

Notice the format of the coordinates: {X, Z, Y}. X controls “side-to-side” position, Y controls “up-and-down” position, and Z controls height above or below the default plane of movement.

So, suppose you want to start each player 51 km to the side of the origin, 9 km above the origin on the Y-axis, and in line with the default plane of movement. You'd type something like this:

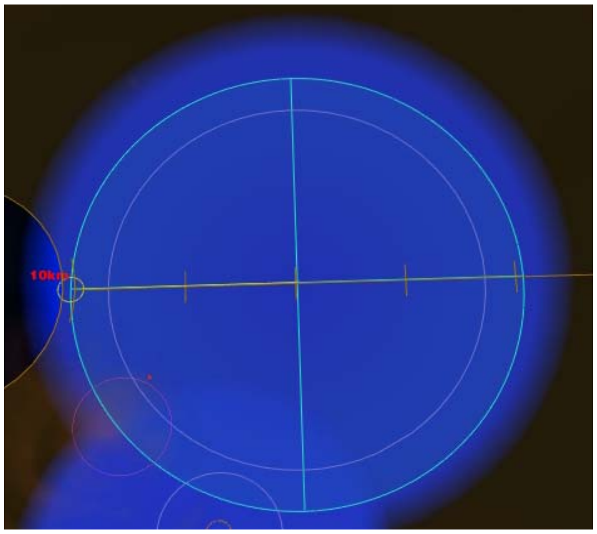
```
function DetermChunk()
  addPoint("StartPos0", {-51000, 0, 9000}
  addPoint("StartPos1", {51000, 0, -9000}
```

Simple enough, eh? Notice that the X and Y values are inverted to keep the playing area symmetrical. A 4-player map using those values might look like this:

```
function DetermChunk()
  addPoint("StartPos0", {-51000, 0,  9000}
  addPoint("StartPos1", {51000,  0, -9000}
  addPoint("StartPos2", {-51000, 0, -9000}
  addPoint("StartPos3", {51000,  0,  9000}
```

Of course, this isn't really symmetrical, since players one and three would be very close to one another, and so would players two and four. But you get the idea.

Keep in mind when setting distances that it takes a scout squad three and a half minutes of uninterrupted travel to move 100 km on conventional drives. That's a pretty long time, and you can expect capital ships to take much much longer when traversing the same distances.



You can measure distance in the Sensors Manager by counting prongs along the movement plate axes. As you can see on the left, the distance between each prong is 5 km by default, or 10.5 seconds of scout traveling time. Plan distances accordingly.

The next bracket set, {A, B, C}, determines the angle of rotation of each player's mothership. Only put in a value for B—A and C should be left at zero (if you change them, they'll tilt the mothership at bizarre angles at the beginning of each map. It will then take 20 or 30 seconds to right itself).

Using B, you can set each player's **starting orientation** (which way his carrier and mothership are facing). If B is set to 90, the player's carrier and MS will hyperspace into the level facing towards 90° on the Sensors Manager compass. Generally it pays to have your ships facing more or less towards the center of the map.

6. Add Resources, Ships, Terrain.

These you can copy and paste into your maps. Just tweak the settings, assign coordinates, and voila—you have *stuff* in your map!

Asteroids

addAsteroid("AsteroidType", {X, Z, Y}, RU, ?, ?, ?, ?)

Default values: *addAsteroid("Asteroid_3", {0.0, 0.0, 0.0}, 100, 0, 0, 0, 0)*

-- AsteroidType ranges from Asteroid_1 to Asteroid_5, with 1 being the smallest and 5 being the largest. Only 3, 4 and 5 are harvestable—1 and 2 are purely ornamental.
-- RU is a multiplier applied to the default RU amount contained by each asteroid type. The larger the asteroid, the more RU it contains by default. An RU value of 100 means 100% of this default RU count, 70 is 70%, etc. A value of 0 means that the asteroid contains no resources, and is purely ornamental.
-- Default asteroid stats: Level 3 asteroids have 9000 RU, level 4s have 18,000 RU, and level 5s have 40,000 RU. Level 3s can support one collector, Level 4s can support two, and Level 5s can support three collectors at a time.

Clouds (I have no clue how this is different from Dust Cloud, but I'd use Dust Cloud instead)

addCloud("?", "CloudType", {X, Z, Y}, {R, G, B, ?}, ?, Size)

Default values: *addCloud("polySurface1", "Cloud_NoRes", {0.0, 0.0, 0.0}, {1, 1, 1, 1}, 0.0, 0.0)*

-- CloudType can be cloud_0, cloud_nores, cloud_nores2 and cloud_nores3.
-- R, G, B is the RGB code (0=0 1=255).
-- Size is the cloud size.

Dust Clouds (these obscure sensors and can be charged with ion beams)

addDustCloud("?", "DustCloudType", {X, Z, Y}, {R, G, B, ?}, ?, Size)

Default values: *addDustCloud("polySurface1", "DustCloud_NoRes", {0.0, 0.0, 0.0}, {1, 1, 1, 1}, 0.0, 0.0)*

-- DustCloudType can be dustcloud_0, dustcloud_nores, dustcloud_nores2, dustcloud_nores2_m05, dustcloud_nores3, dustcloud_nores3_m05, dustcloud_nores_m05, dustcloud_nores_nocharge and dustcloud_teal.
-- R, G, B is the RGB code (0=0 1=255).
-- Size is the dust cloud size.

Megaliths (these are huge; they make great map props!)

addSquadron("SquadronName", “SquadronType,” {X, Z, Y}, ?, {?, ?, ?} ?, ?)

Default values: *addSquadron("meg_ asteroid", "meg_asteroid", {0, 0, 0}, -1, {0, 0, 0}, 0, 0)*

-- As far as I can tell, SquadronName and SquadronType can both be set to the same thing and will work fine. Make sure you put something in for both, however, or else the game won't recognize it.
-- SquadronName and SquadronType can be any of the following:

meg_asteroid, meg_asteroid_nosubs, meg_asteroid_mp, meg_balcoragate, meg_bentus, meg_bentus_ruin_1 (up to 11), meg_bentus_ruined, meg_bentus_ruins_core_1 (up to core_3), meg_bigred, meg_chimera, meg_dreadnaughtberth, meg_foundry, meg_foundrydebris_chunk1 (up to 4), meg_foundrydebris_flake1 (up to 4), meg_gehenna_1 (up to 7), meg_misslefrigate (yes, spelled like that), meg_progenitorpowermodule, meg_progenitorpowertrigger, meg_progenitorpowertrigger_noui, meg_sajhulknose, meg_sajhulknose_ui, meg_sajhulkpanels, meg_sajhulkpanels_ui, meg_sajhulkturret, meg_salvagecollector, meg_tanis, meg_tanisstructure_medium, meg_tanisstructure_medium2, meg_veildebris_bit1 (up to 15), meg_veildebris_chunk1 (up to 5; alternate versions have "_nd" on the end), meg_veildebris_chunk_lighthouse, meg_veildebris_flake1 (up to 10).

Nebulae (these damage ships!)

addNebula("?", "NebulaType", {X, Z, Y}, {R, G, B, ?}, ?, Size)

Default values: *addNebula("Nebula4_OLD1", "M11_Bentusi_Radiation", {0.0, 0.0, 0.0}, {1, 1, 1, 1}, 0.0, 0.0)*

-- NebulaType can be m05_dustcloud_nebula, m05_nebualdustcloud_nores, m05_nebualdustcloud_nores2, m05_nebualdustcloud_nores3, m07_foundry_radiation, m08_nodamage_radiation, m11_bentusi_debris, m11_bentusi_radiation, mp_bentusi_radiation, nebula01_cream, nebula01_teal, nebula_0, nebula_hiding and radiation.
-- R, G, B is the RGB code (0=0 1=255).
-- Size is the nebula size.

Pebbles (the little brown specks which show up in sensors manager but do nothing)

addPebble("PebbleType", {X, Z, Y}, ?, ?, ?)

Defaults: *addPebble("Pebble_0", {0.0, 0.0, 0.0}, 0, 0, 0)*

-- PebbleType can be pebble_0, pebble_1, pebble_2 or pebble_3 .

Salvageable Debris

addSalvage("ChunkType", {X, Z, Y}, RU, ?, ?, ?, ?)

Default values: *addSalvage("Slv_Chunk_Lrg03", {0.0, 0.0, 0.0}, 100, 0, 0, 0, 0)*

-- ChunkType ranges from Slv_Chunk_Lrg01 to Slv_Chunk_Lrg05, with 1 being the smallest and 5 being the largest. You can also put in Slv_Chunk_Sml01 through to 08.

-- RU is a multiplier applied to the default RU amount contained in each debris chunk. This works the same as with asteroids.

Ships

addSquadron("ShipName", “ShipName,” {X, Z, Y}, F, {A, B, C} ?, ?)

Default values: *addSquadron("vgr_shipyard", "vgr_shipyard", {0, 0, 0}, -1, {0, 0, 0}, 0, 0)*

-- Use A, B and C to rotate the ship. Some ships (like vgr_planetkiller) will need to be rotated. In the case of the planetkiller, set its A value to 270.
-- F determines who controls it. If F is -1, no one controls it and it is capturable. If it is set to 0, player[0] controls it, if it is set to 1, player[1] controls it, etc.
-- ShipName can be any of the following (I’m not sure if they all work, though):

Hiigaran: hgn_assaultcorvette, hgn_assaultcorvetteelite, hgn_assaultfrigate, hgn_attackbomber, hgn_attackbomberelite, hgn_battlecruiser, hgn_carrier, hgn_cloakingfrigate, hgn_defensefieldfrigate, hgn_destroyer, hgn_dreadnaught, hgn_drone_frigate, hgn_drone_frigate_2, hgn_drone_frigate_3, hgn_ecmprobe, hgn_gunplatform, hgn_gunturret, hgn_hscore, hgn_interceptor, hgn_ioncannonfrigate, hgn_marinefrigate, hgn_marinefrigate_soban, hgn_minelayercorvette, hgn_mothership, hgn_probe, hgn_proximitysensor, hgn_pulsarcorvette, hgn_pulsarplatform, hgn_resourcecollector, hgn_resourcecontroller, hgn_scout, hgn_shipyard, hgn_shipyard_elohim, hgn_shipyard_spg, hgn_supportfrigate, hgn_targetdrone, hgn_torpedofrigate.

Vaygr: vgr_artillerycruiser, vgr_assaultfrigate, vgr_battlecruiser, vgr_bomber, vgr_carrier, vgr_commandcorvette, vgr_commstation, vgr_destroyer, vgr_heavymissilefrigate, vgr_hyperspace_platform, vgr_infiltrator_frigate, vgr_interceptor, vgr_lancefighter, vgr_lasercorvette, vgr_listeningpost, vgr_minelayercorvette, vgr_missilecorvette, vgr_mothership, vgr_mothership_makaan, vgr_planetkiller, vgr_planetkillermissile, vgr_prisonstation, vgr_probe, vgr_probe_ecm, vgr_probe_prox, vgr_resourcecollector, vgr_resourcecontroller, vgr_scout, vgr_shipyard, vgr_transportfrigate, vgr_weaponplatform_gun, vgr_weaponplatform_missile.

Keeper: kpr_attackdroid, kpr_destroyer, kpr_destroyerm10, kpr_mover, kpr_mover_capture, kpr_mover_salvage, kpr_sajuuk, kpr_sajuuk_nosensors.

7. Size the map.

Now set the size of your map:

```
  setWorldBoundsInner({0, 0, 0}, {X, Z, Y})
end
```

Don’t change the values in the first bracket! Those effect position, and may give you a lopsided map if changed from 0. The second bracket has the X, Z, and Y size values you want.

8. Add Map Info.

Once you've set the size, you have gotten to the "NonDetermChunk" part of the map. This consists of: setting the background image, picking the music, choosing shadow color, deciding how far out and in you can zoom the sensors manager camera, and picking whether you want fog or a lens flare. Add this text:

```
function NonDetermChunk()

  fogSetActive(0)
  setGlareIntensity(0)
  setLevelShadowColour(0, 0, 0, 1)
  loadBackground("X")
  setSensorsManagerCameraDistances(MIN, MAX)
  setDefaultMusic("Data:sound/music/Y/Z")
end
```

Values for X can be any of the following: black, white, m01, m02, m03, m04, m05, m06, m07, m08, m09, m10, m11, m12, m13, m14 or m15.

Y is one of the music directories, either "ambient" or "battle." Z is the track name.

Tracks in the **Ambient** directory include: amb_01, amb_02, amb_03, amb_04, amb_05, amb_06, amb_07, amb_08, amb_09, amb_10, amb_11, amb_12, amb_13 and amb_14.

Most of the ambient tracks are pretty boring, honestly, but if you want a relaxed atmosphere, they can work.

Tracks in the **Battle** directory include: battle_01, battle_04, battle_04_alt, battle_06, battle_keeper, battle_movers, planet_killers, Battle_sajuuk and bentus_arrival.

Planet Killers is pretty dramatic. Battle Keeper, Battle_Movers, Bentus Arrival and Battle Sajuuk are pretty cool Arabic-sounding tracks. Battle 01, Battle 04 and Battle 06 are your typical menacing orchestral tracks.

9. Extra stuff:

To add a **level screenshot**: add an uncompressed 24 (or 32)-bit .tga with the same name as your map to ..\Data\ui\mapthumbnails\multiplayer\deathmatch.
Tip: the game will only display the top-left portion of the image, so crop accordingly.
Tip: the final image must be at least 128 by 128 pixels, or some multiple thereof (256 by 256, 512 by 512, etc.) 128 by 128 will look blurry; 256 by 256 looks much better.
Tip: the game will not display the image if your map file has an extra period in it. For example, Big_Map_2.0.tga will not display, whereas Big_Map_v2.tga will. Name your .level file and .tga accordingly.

HW1 to HW2 **map convertor**, provided by ZuilJin:
http://forums.relicnews.com/showthread.php?s=&threadid=18219

I hope you found this helpful! Thanks to ZuilJin for first finding so much of that nifty map code and telling us about it, and to EvilChekov for providing me with a guinea pig map.

-Malignus (craigstern@hotmail.com)