Soldier of Fortune II
Random Mission Generator
Tutorial/Technical Information

How does the random mission generator work? The short and simple answer is: The Random Mission Generator (RMG) carves some random paths into a large piece of terrain and then takes a pile of pre-made buildings and enemies and scatters them randomly onto the paths. Of course, in reality it's not quite that random, since we have the ability to control how it draws the paths and what it puts on the paths and where. The RMG takes two main files: the **.instance** file and the **.mission** file and uses the information to determine how to lay out the map.

INSTANCE FILES

An instance file is basically a big list of instances and their associated properties.

What is an "instance"? An instance is anything that is *placed* into a randomly generated map, such as enemies, buildings, bridges, trees, villages, clearings, etc.

There are 5 kinds of instances that the RMG currently knows about:

void – This is the most generic of all instances, it doesn't really contain anything interesting, but it's useful for making flat or empty areas in the map.

bsp – This is the most common of all instances. It points to a .bsp file, which is then read in and placed in the map. The .bsp instances can be anything you can make in a level editor, with some exceptions and limitations:
-They must be relatively small in both file size and poly count (A good rule of thumb is keep them within 1024x1024 units in size)
-They must be centered around 0,0,0
-You will usually want to have a deep foundation on them. (64 units should suffice)
-Use auto-closing doors and area portals to make the insides of buildings not get drawn when the doors are closed. NOTE: Doors do not exist in multiplayer RMG.
-Use **misc_model**s whenever possible instead of **model_static.** This will help keep the number of entities in the map to a minimum. Misc models should have

the RMG flag checked and the shaders that they use should have _BSP_RMG at the end of their name and be flagged “lightingDiffuse”
-The shaders on any outdoor surfaces that would be lit by the sun (or moon) need to have the following parameters set on them:

    q3map_nolightmap
    q3map_onlyvertexlighting
    rgbGen lightingDiffuse

See the shaders tutorial if you don’t understand what this means.
-No single player style breakable glass
-All brush entities **MUST** have *origin* brushes associated with them
-There’s a whole slew of other limitations that the RMG holds over normal single player levels, but the ones listed above are the most important to follow.

npc - These are your enemies they point to a classname of a pre-defined npc. See the NPC template tutorial to find out how to make your own enemies, or use one of SOF2’s many various baddies.

random – This is what you can use if you want to have a group of things that get randomly chosen between. For example, you can set up a random instance called “tree” and have it randomly choose one of several different tree instances wherever you want a “tree” placed.

group – Use this to create groups of instances to form villages or squads of enemies.

FILE FORMAT

The following is an example of what you would see in an .instance file. Anything that appears in blue is a comment that does not get used by the engine.

```
instances
{
        void
        {
                name                                    “small_clearing”
                        // name can be just about anything you want.  Avoid names with spaces
                spacing                         256
                        // Imagine a circle with a radius of 256.  No other instances will be found
        within that circle.
                flatten                         128
                        //  Flattens a circular area with radius 128
                        // This parameter is optional
                surfacesprites                  yes
                        // Put this value in if you want grass, bushes, etc to grow within the
        flattened area.  Only multiplayer RMG recognizes this parameter.
        }       // end of void instance
```

```
bsp
{
        name                                    “building1”
                // the name of an instance does not have to match the filename of the
bsp file
        file                                    “instances/generic/building01”
                //  Points to a .bsp file
        spacing                                 800
                // Acts the same way as it does for void instances.  Generally you want to
have the spacing encompass all of your geometry in the .bsp file with a little extra buffer
space.  This will prevent buildings from overlapping when placed next to each other.
        flatten                                 128
                // Again, same deal as void flatten.  Make sure you flatten an area big
enough for your building to fit onto.
        baseangle                                       90
                // The RMG has the capability to make instances face in specific
directions.  For example, you may want a tower on a path to a village to face away from
the village for protection, or you may want all the buildings in the village to face the
center.  However, not all bsp instances are constructed with the same alignment.
Baseangle allows you to specify a base rotation for the building to get it to face the right
way.
        automap_symbol                                  “none”
                //  When the player looks at his automap, he will see little boxes for
buildings, circles for objectives, triangles for starting points and X marks the spot of the
exit point.  Every instance needs a value for automap symbol.  Valid values are:  “none”,
“start”, “end”, “building”, “objective”, “friend”, and “enemy”.  The friend and enemy values
are currently only used for debugging purposes.
        anglevariance                           180
                //  This optional parameter is an easy way to make all of your instances
face in random directions.  Useful for small cover, trees and other instances that get used
a lot.
        surfacesprites                          yes
                // Put this value in if you want grass, bushes, etc to grow within the
flattened area.  Only multiplayer RMG recognizes this parameter.
        objective_message                       "RMG_COL_DEMO1_MSG"
                // Only for objective instances. Single player only – this can be a token
pointing to a value in a .sp file or plain text
        objective_description       "&RMG_COL_DEMO1_DESC&"
                // Only for objective instances. Single player only – this can be a token
pointing to a value in a .sp file or plain text

        objective_info                          "&RMG_COL_DEMO1_INFO&"
                // Only for objective instances.  Single player only – this can be a token
pointing to a value in a .sp file or plain text

}       // end of bsp instance


npc
//  npcs can only be used in single player RMG missions.
{
        name                    "sergeant"
                // RMG name for the npc type
        classname               NPC_Colombian_Sergeant
                // name of the npc entity used in the game
        spacing                 32
            // spacing around the npc
        area_point_goal         "exit"
            // This optional and rarely used value will make the npc immediately
    begin moving to the area point named. NOTE: This value should only be used on NPCs
    with sergeant ranking.  Sergeants will automatically find team mates near them and
    recruit them to go with him to the area point.
        automap_symbol          "enemy"
            //  Generally for npcs, you're only going to use friend or enemy values
        surround_goal_dist      "1500"
            // Another optional and rarely used value.  This is how far you must be
    away from an npc for him to try to surround you.
        objective               "assassinate"
            // This value is only to be used on special npcs who are related to an
    objective in the mission.
    }   // end of npc instance


    random
        // random instances can contain as many different instances as you would like
    {
        name                    "tree"
            // when you place a "tree" instance into a randomly generated map, you
        will get only one of the following instances:

        instance
        {
            name                "tree1"
                // this must be the name of an instance defined somewhere
    above this point in the file.
        }

        instance
        {
            name                "tree3"
                // pretend this instance has been previously defined in the file
            multiplier          3
                // Makes this instance 3 times more likely to happen.  NOTE:
        multiplier only works in multiplayer RMG.
        }
        instance
        {
            name                "tree4"
                // pretend this instance has been previously defined in the file
            multiplier          2
        }

    }   // end of random instance


    group
        // groups can contain any number of other instances.  All instances named in the
group must have been previously defined in the .instance file.  One thing to note about groups is
that entities are placed in the order they are listed.  So the last entity listed will be placed last.
This has the effect of entities that are placed later push other entities out and away from them.
    {
        name                    "small_village"
                // the name of this group
        padding                 10
                // every instance in the group will be spaced a minimum of 10 units away
from every other instance.  This value is in addition to the spacing defined on the
instances themselves
        confine                 1800
                // Keep all the instances in this group within an overall radius of 1800
units
        instance
        {
                name                    "large_flatten"
        }

        instance
        {
                name                    "building"
                        // in this case, building points to a random instance containing a
                list of all the different buildings
                mincount                1
                        // minimum # of buildings in the village (optional, defaults to 1)
                maxcount                2
                        // maximum # of buildings to appear in the village (optional,
        defaults to 1)
                minrange                100
                        // minimum distance from the center of the group to place this
        instance (optional, defaults to 0)
                maxrange                500
                        // maximum distance from the center of the group to place this
        instance (optional, defaults to 0)
                anglevariance   45
                        // all instances in a group default to orienting themselves toward
        the center of the group.  With this angle variance, the instance will randomly
        rotate itself up to 45 degrees in either direction
        }

        instance
        {
                name                    "smallcover"
                        // pretend this instance has been previously defined in the file
                mincount                2
                maxcount                3
                minrange                100
                maxrange                500
                anglevariance   45
        }

        instance
        {
                name                    "vaultcover"
                        // pretend this instance has been previously defined in the file
                mincount                1
                maxcount                2
                minrange                100
                maxrange                500
                anglevariance   45
```

```
		}

		instance
		{
			name			"pickups"
				// pretend this instance has been previously defined in the file
			mincount		2
			maxcount		3
			anglevariance	180
		}

		instance
		{
			name			"random_squad"
				// pretend this instance has been previously defined in the file
			minrange		10
			maxrange		200
			mincount		2
			maxcount		2
		}

		instance
		{
			name			"flagpole"
				// pretend this instance has been previously defined in the file
		}
	}	// end of group instance

}	// end of instance file
```

Designing RMG Maps

At this point, you may be thinking, “Why do I need to *design* a random map? Isn’t it supposed to be random?” In actuality, completely random maps don’t play very well, they can contain large, boring sections and other sections can end up overpopulated with architecture and npcs, causing the game engine to grind to a halt. In SOF2, the randomly generated maps are tightly controlled, to keep the engine running at a reasonable rate, yet set up to give the player a wide variety of maps within a mission type.

NODES

First, you need to decide how many nodes you need to accomplish the mission that you have in mind. A node is a point of interest on a map, such as an objective location, village, starting point, finishing point, etc. The first thing that the RMG does when generating a map is divide the terrain into a number of nodes that you specify in the mission file. The **x_cells** and **y_cells** values let you define a grid of nodes. This grid is not an evenly divided grid with each node equidistant from the next. The nodes are jittered a bit to make the map be a little less regular, however, the more nodes you have, the closer together they will be – which gives you less room to work with for each node. Most of SOF2s single player maps take place on a 3x3 grid of nodes and have 6 or less nodes actually used on the grid.

PATHS

Think of the paths getting carved into a large block of terrain. First, the RMG picks a start node (depth 0) and draws a random path to the deepest depth node. This path may be straight, or it may twist and turn over itself. Once the longest path is drawn, the engine then picks the second deepest node and attempts to fit it onto the existing path. If the RMG can't fit the node onto the existing path because there's already a used node in the place it's trying to pick, or because of **min_paths** or **max_paths** constraints, it will create an offshoot path to an empty node. This continues until all the defined nodes have a home. Sometimes you'll get a node that is off on it's own with no paths leading to it. This happens when the engine can't find create a path to it, due to min/max_path constraints. If this happens, loosen your constraints and try it again.

PLACING INSTANCES

Instances can be placed in two ways: directly on a node, or on paths that are leading to and from a node.
You can place as many instances on a node as you like, but keep in mind that each additional instance will automatically push previously placed instances further out away from the node. If you wish to have a village placed at a node, it is best to create a group instance using the *confined* attribute and place the group instance on the node, rather than placing each building individually.
When instances are placed on paths using the *instanceonpath* command, they will be placed on **all** paths connected directly to that node. So, if you have two adjacent nodes and you place instances on the paths leading to each of them, the path between the two nodes will be doubly populated.

TIPS FOR DESIGNING PATHS

There are two basic schools of RMG design that we have found to be useful here at Raven Software: The ultra-controlled single player style of design and the less-controlled multiplayer style of design. For single player, the player's fun relies heavily on how the map is set up due to AI restrictions, mission objectives and such, so we opted to make the RMG maps more structured. Whereas for multiplayer, the map is merely a backdrop for everyone to use in whatever ways they see fit, so we went with less controlled maps to allow a wider variety of maps to occur. There are some other major differences between multiplayer and single player that affected the ways we set maps up such as the introduction of symmetrical maps in multiplayer, but I'll get into those details later.

SINGLE PLAYER TIPS
If you examine the node “placements” in any of our single player maps, you’ll notice that every node is basically assigned a depth – the minimum depth is the same as the maximum depth. Nodes are not allowed to “float” around since we want objectives to be spaced at specific distances for game pacing. This does not mean that all maps will look exactly alike, but it does mean that they will all have similarities. To add variety, we often added extra nodes in at the same depth that major objectives are at so that sometimes the major objectives would be on the main path and sometimes they would be off on a spur path. When it came to populating the paths, we usually only used the *instanceonpath* directive on every other node, rather than every single node. This way, we avoided double populating and sections of path, which made it easier to debug problems and tweak specific sections of path since all the path population information is nicely bundled in one area.

MULTIPLAYER TIPS
Multiplayer maps are set up a bit different. Here we have the ability to make maps symmetrical which changes the way things work quite a bit. Basically, the grid of nodes is split diagonally down the center and the node at depth 0 is in a far corner. The paths are then created normally. When instances are placed, anything on the opposite side of the center line is removed and anything near the center is bumped back. Once all the paths are created and instances are placed, everything is mirrored on the other side of the center line. With multiplayer maps, we have more “floating” nodes by having different *min_depth* and *max_depth* values. Instances are also placed with fewer restrictions since we use *instanceonpath* on every single node. Note that we place half as many instances on the paths associated with each node as we do in single player to keep the paths from becoming over populated

MISSION FILES

Mission files define the layout of the paths and nodes in the map and place entities that have been defined in the instance files.

SINGLE PLAYER MISSION FILE FORMAT

Here is part of a single player mission with comments describing the various fields. I’d recommend taking a look at the complete mission files used in the single player game for a more complete view on how things work.

```
mission
```

```
{
    description                 "&RMG_ASSASSINATIONDESC&"
        // description of the mission – can point to a token in a .sp file or be plain text
    exitScreen                  rmg_assassination
        // screen to show on completion
    TimeExpiredScreen           rmg_timeexpired
        // screen to show on failure

    easy
        // mission set up for easy difficulty
    {
        timelimit                   20
            // time limit for missions with time limit activated
        health                      100
            // starting health for player
        armor                       100
            //starting armor for player
        npcaccuracy                 0.9
            // modifier for all npc's accuracy (on top of built in difficulty adjustments
made by the engine)
        npchealth                   0.9
            // modifier for all npc's health (on top of built in difficulty adjustments
made by the engine)
        pickup_health               0.5
            //  percentage of health pickups flagged RMG to activate  (50%)
        pickup_armor                0.5
            //  percentage of armor pickups flagged RMG to activate  (50%)
        pickup_ammo                 0.5
            //  percentage of ammo pickups flagged RMG to activate (50%)
        pickup_weapon               0.5
            //  percentage of weapon pickups flagged RMG to activate (50%)
        pickup_equipment            0.5
            //  percentage of equipment pickups flagged RMG to activate (50%)


        outfit
            // this section contains the weapons that will be automatically outfitted for
the player if they choose not to select their own weapons
        {
            random_choice
                // first possible weapon set out of 2 for this difficulty level
                // random choice allows you to have multiple options for just
            about anything in the RMG mission files
            {
                random_weight 4
                    // this random choice will occur 4 times as often

                weapons
                    // these are the weapons to give to the player.
                    // the numbers after the weapon represent weapon,
    ammo in clip, accessories and accessory ammo in clip.  It's best to leave the numbers as
    they appear in the mission files since much of the weapon system is hard-coded into the
    engine and something may break if you try to do something fancy.  ☺
                {
                    "Knife"         "1/1/1/0"
                    "US SOCOM"              "1/12/0/0"
				"MSG90A1"			"1/5/0/0"
				"M4"				"1/30/1/0"
			}

			ammo
				// give the player this for ammo
			{
				"5.56mm"			"150"
					// ammo for M4
				"40mm HE grenade"	"6"
					// ammo for M4 grenade launcher
				"0.45 ACP"			"48"
					// ammo for socom
				"7.62mm"			"25"
					// ammo for MSG90A1
				"Knife"				"6"
			}
		} // end of first possible weapon set


		random_choice
			// second possible weapon set for this difficulty level
		{
			random_weight 1
				// this weapon set gets chosen considerably less since
			it's weight is only 1
			weapons
			{
				"Knife"				"1/1/1/0"
				"M1911A1"			"1/7/0/0"
				"Binoculars"		"1/0/0/0"
				"M3A1"				"1/30/1/0"
			}

			ammo
			{
				"0.45 ACP"			"90"
				"Knife"				"6"
				"M67"				"5"
			}
		} // end of second possible weapon set
	} // end of outfitting section

	objectives
		// these are all of the objectives that the player will encounter
	{
		leader
		{
			trigger			"assassinate"
				// targetname of a trigger objective that gets used when
this objective is completed
			message				"RMG_ASSASSIN01"
				// message that gets printed when objective is complete
			description		"&RMG_ASSASSIN02&"
				// description of the objective
			info			"&RMG_ASSASSIN05&"
```

```
                        // additional information for the objective
                completed_sound “sound/misc/rmg/objective_complete"
                        // sound that is played when objective is completed
        }

        helicopter
        {
                trigger         "helicopter"
                priority  10
                message                 "RMG_DEMOLITION05"
                description     "&RMG_ASSASSIN03&"
                info            "&RMG_ASSASSIN04&"
                completed_sound "sound/misc/rmg/objective_complete"
        }
} // end of objectives section



nodes
        // This is the section that determines the layout of the paths/canyons
        // The information presented here is strictly explanatory for field values
        // For detailed information on layout schemes in the single player RMG,
        // See the section titled “Designing RMG maps” which appears
        // elsewhere in this document
{
        x_cells         3
        y_cells         3
                // here we are splitting up the terrain on the map into a 3x3 grid
of nodes (9 nodes maximum)  Each node represents a possible “interest point” on the map

        start
                // This is the name of the node, you can call it anything as long
as it has no spaces.
                // eventually this node will contain the spawn spot for the player
        {
                min_depth 0
                // minimum depth of this node
                max_depth 0
                // maximum depth of this node
                min_paths 1
                // minimum number of paths leading to this node (default 1)
                max_paths 2
                // maximum number of paths leading to this node (default 1)
        }

        exit
        // eventually this node will contain the extraction point
        {
                min_depth 3
                max_depth 3
        }

        center
        // eventually this node will contain a village
        {
                min_depth 2
                max_depth 2

                min_paths 1
                max_paths 3
            }

            oddball
            // this node is just an extra place on the map to make it more interesting
            {
                min_depth 1
                max_depth 1
                min_paths 1
                max_paths 3
            }


            dead_end
            // notice that this node is at the same depth as the center node.  By
having 2 nodes at the same depth, we alter the location of the village node so it’s not always on
the main path.
            {
                min_depth 2
                max_depth 2
            }
        } // end of node definitions

        rivers
        // rivers are optional.  This entire section can be omitted
        // there are 3 possible choices for rivers here: normal rivers, shallow rivers and
    no rivers.
        {
            random_choice
            {   // normal river
                maxpathdepth 6
                // this is the length of the river in nodes (rivers do not use the
            same nodes as the paths – river nodes are placed between the path
            nodes automatically
                depth       .1
                // depth is on a scale from 0 - 1.
                minwidth    .03
                // the value provided for minwidth represents a percentage of the
surface area of the map.  Here we are defining the minimum width of any part of the river to be
3% of the width of the entire map
                maxwidth    .04
                // the value provided for maxwidth represents a percentage of
the surface area of the map.  Here we are defining the maximum width of any part of the river to
be 4% of the width of the entire map
                points      8
                // How many nodes should be placed for the river
                deviation   .03
                // How “wiggly” is the river – higher values mean more wiggles
                breadth     7
                // The easiest way to describe this is that it’s a “steepness factor”
of the banks of the river
                bridge      "bridge"
                // name of the instance  (can be a random instance) that is used
when a bridge is automatically determined to be needed.
            }
            random_choice
            {   // shallow river
                maxpathdepth 6
                depth           .15
                minwidth        .03
                maxwidth        .04
                points          8
                deviation       .03
                breadth         6
                bridge          "bridge"
            }
            random_choice
            {   // no river
                random_weight 2
                // since this choice is weighted 2 and the other 2 are only
weighted 1, that makes 4 total weights, so this choice will be chosen 50% of the time.
                depth           1
            }
        } // end of river definitions

        paths
        // this defines the attributes of the paths – the values here have the same
meanings as they do for the rivers
        {
            random_choice
            {
                depth           .25
                minwidth        .025
                maxwidth        .04
                points          10
                deviation       .03
                breadth         7
            }
            random_choice
            {
                depth           .25
                minwidth        .03
                maxwidth        .04
                points          8
                deviation       .03
                breadth         6
            }
        } // end of path definitions

        instances
        // this is the section where we define which instances we want and where we
want them
        {
            allstart
            // allstart is the name of the spawn spot instance
            {
                origin
                {
```

```
                                node                    "start"
                                // this section defines which node to place allstart at
                        }

                        instanceonpath
                        // instance on path will place 0 or more instances on every path
connecting to the node defined above.  The node named, “start”, in this case.
                        {
                                instance                "tree"
                                // “tree” must be defined in the .instance file
                                minposition 0.05
                                // minposition and maxposition define what sections of
the path contain this instance.  Think of the value as being a percentage along the path
from this node to the next node. The value must be between 0-1.  0 is always towards
node 0.
                                maxposition 0.8
                                // maximum position of this instance along the path
                                mincount                3
                                // minimum number of this instance to appear on each
                path it’s placed.  NOTE: all instances defined in a single
                “instanceonpath” will be spaced semi-evenly.
                                maxcount                7
                                // maximum number of this instance to appear on each
        path it’s placed.
                        }


                        instanceonpath
                        {
                                instance                "naturalcover"
                                minposition 0.3
                                maxposition 0.8
                                mincount                2
                                maxcount                3
                        }

                        instanceonpath
                        {
                                instance                "vaultcover"
                                minposition 0.3
                                maxposition 0.8
                                mincount                0
                                maxcount                1
                        }

                        instanceonpath
                        {
                                instance                "smallcover"
                                minposition 0.4
                                maxposition 0.8
                                mincount                1
                                maxcount                2
                        }


                        instanceonpath
```

```
		{
			instance		"random_squad"
			minposition 0.4
			maxposition 0.7
			mincount 1
			maxcount 2
		}

		instanceonpath
		{
			instance		"random_defense"
			minposition 0.6
			maxposition 0.6
			pathalign 0.5
			mincount 0
			maxcount 1
		}

	}

	commando_team
	{
		origin
		{
			node			"oddball"
		}
	}

	largecover
	{
		origin
		{
			node			"oddball"
		}
	}


	village1
	{
		objective			"leader"

		origin
		{
			node			"center"
		}

		instanceonpath
		{
			instance		"tree"
			minposition 0.3
			maxposition 0.7
			mincount		3
			maxcount		6
		}

		instanceonpath
		{
			instance		"naturalcover"
			minposition 0.3
			maxposition 0.7
			maxcount		2
		}

		instanceonpath
		{
			instance		"smallcover"
			minposition 0.4
			maxposition 0.6
			maxcount		2
		}

		instanceonpath
		{
			instance		"vaultcover"
			minposition 0.4
			maxposition 0.6
			mincount		1
			maxcount		2
		}


		instanceonpath
		{
			instance		"random_squad"
			minposition 0.4
			maxposition 0.7
			mincount		2
			maxcount		2
		}

		instanceonpath
		{
			instance		"random_defense"
			minposition 0.55
			maxposition 0.55
			pathalign 0.5
		}



	}

	bunker
	{
		origin
		{
			node			"dead_end"
		}

		instanceonpath
		{
			instance		"tree"
		minposition 0.2
		maxposition 0.8
		mincount	3
		maxcount	6
	}

	instanceonpath
	{
		instance	"smallcover"
		minposition 0.4
		maxposition 0.6
	}

	instanceonpath
	{
		instance	"vaultcover"
		minposition 0.4
		maxposition 0.6
		mincount	0
		maxcount	1
	}

	instanceonpath
	{
		instance	"naturalcover"
		minposition 0.3
		maxposition 0.7
		mincount	1
		maxcount	2
	}

	instanceonpath
	{
		instance	"largecover"
		minposition 0.4
		maxposition 0.6
		mincount 0
		maxcount 1
	}

	instanceonpath
	{
		instance	"smallpatrol"
		minposition 0.7
		maxposition 1.0
		mincount 0
		maxcount 2
	}
}

helicopter
{
	objective				"helicopter"
	hide_until_objective_complete	"leader"

	origin
```

```
                        {
                                node            "exit"
                        }
                    }
                } // end of instances section

            } // end of easy definition
```

```
    superhard
    // this defines the attributes for the game on “superhard difficulty”.  It’s possible to define
completely different settings for different difficulties.  You should also define medium and hard
difficulty levels, even if they only inherit “easy”.  It is also possible to inherit attributes from a
difficulty level that is inheriting it’s attributes from a different difficulty level.
    {
        inherit         "easy"
        // inherit takes all the attributes from the easy mission and give them to
superhard.  Anything defined below will override the easy settings

        health              100
        armor               100
        timelimit           22
        npcaccuracy         0.99
        npchealth           1
        pickup_health       0.3
        pickup_armor        0.3
        pickup_ammo         0.1
        pickup_weapon       0.1
        pickup_equipment    0.1

        outfit
        {
            weapons
            {
                "Knife"             "1/1/1/0"
                "US SOCOM"          "1/12/0/0"
                "MSG90A1"           "1/5/0/0"
            }

            ammo
            {
                "0.45 ACP"          "24"
                "Knife"             "6"
                "7.62mm"            "15"
            }
        } // end of outfitting section

        instances
        // instances defined here will be IN ADDITION TO the instances defined in the
inherited game definition
        {
            osprey
            {
                hide_until_objective_complete    "leader"
                // hide_until_objective_complete will make this instance invisible
    and inactive until the “leader” objective is completed.  This is useful for spawning in
    additional enemies or objectives later in the game.
                origin
                {
                    node            "exit"
                }
            }

            tough_guy
            {
```

```
                                        origin
                                        {
                                                  node                "oddball"
                                        }
                              }
                    } // end of instances section
          } // end of superhard definition
}// end of mission file
```

MULTIPLAYER MISSION FILE FORMAT

Here is a multiplayer mission with comments describing the various fields. The format is almost identical as the single player format, except that some sections from single player are not used and there are a few extra parameters. I'd recommend taking a look at the complete mission files used in the multiplayer game for a more complete view on how things work.

```
mission
{
          description                    "CTF"
          //  short description of the gametype
          symmetric                      1
          // leave this field out if you don't want a symmetric map
          padding                        50
          // this pads all instances with 50 units on top of the padding defined for them in the
.instance file

          standard
          // since there is no difficulty level in multiplayer, we use standard
          {
                    nodes
                    // node definitions, just like in single player.  Note the nodes can float to different
depths on the path.  NOTE: for symmetric maps, it is a good idea to keep the number of defined
nodes around half the number total nodes.  The number of placed nodes actually doubles once
the map is mirrored.
                    {
                              x_cells             3
                              y_cells             3

                              team_start
                              {
                                        min_depth 0
                                        max_depth 0
                                        min_paths 1
                                        max_paths 3
                              }

                              dead_end1
                              {
                                        min_depth 1
                                        max_depth 2
```

```
		min_paths 1
		max_paths 8
	}

	dead_end2
	{
		min_depth 2
		max_depth 3
		min_paths 1
		max_paths 8
	}

	dead_end3
	{
		min_depth 1
		max_depth 2
		min_paths 1
		max_paths 8
	}


	supply
	{
		min_depth 1
		max_depth 2
		min_paths 1
		max_paths 3
	}
} // end of nodes

paths
{
	random_choice
	{
		depth		.19
		minwidth	.04
		maxwidth	.055
		points		8
		deviation	.03
		breadth		7
	}

	random_choice
	{
		depth		.19
		minwidth	.039
		maxwidth	.05
		points		10
		deviation	.025
		breadth		7
	}
} // end of paths

instances
{
	cover_clump
```

```
{
	origin
	{
		node		"dead_end1"
	}

	instanceonpath
	{
		instance	"smallcover"
		minposition 0.4
		maxposition 0.6
		mincount 1
		maxcount 2
	}

	instanceonpath
	{
		instance	"naturalcover"
		minposition 0.3
		maxposition 0.7
		mincount 1
		maxcount 3
	}

	instanceonpath
	{
		instance	"tree_clump"
		minposition 0.2
		maxposition 0.8
		mincount 2
		maxcount 4
	}
}

cover_clump
{
	origin
	{
		node		"dead_end3"
	}

	instanceonpath
	{
		instance	"smallcover"
		minposition 0.4
		maxposition 0.6
		mincount 1
		maxcount 2
	}

	instanceonpath
	{
		instance	"naturalcover"
		minposition 0.3
		maxposition 0.7
		mincount 1
```

```
			maxcount 3
		}

		instanceonpath
		{
			instance		"tree_clump"
			minposition 0.2
			maxposition 0.8
			mincount 1
			maxcount 2
		}

		instanceonpath
		{
			instance		"tower"
			minposition 0.5
			maxposition 0.5
			pathalign		0.55
			mincount 0
			maxcount 1
		}
	}

	large_clearing
	{
		origin
		{
			node			"dead_end2"
		}

		instanceonpath
		{
			instance		"smallcover"
			minposition 0.4
			maxposition 0.6
			mincount 1
			maxcount 2
		}

		instanceonpath
		{
			instance		"naturalcover"
			minposition 0.3
			maxposition 0.7
			mincount 1
			maxcount 3
		}

		instanceonpath
		{
			instance		"tree_clump"
			minposition 0.2
			maxposition 0.8
			mincount 2
			maxcount 4
		}
		instanceonpath
		{
			instance		"mp_random_defense"
			minposition 0.5
			maxposition 0.5
			pathalign		0.55
			mincount 0
			maxcount 1
		}
	}

	supply_village
	{

		origin
		{
			node			"supply"
		}

		instanceonpath
		{
			instance		"smallcover"
			minposition 0.4
			maxposition 0.6
			mincount 1
			maxcount 2
		}

		instanceonpath
		{
			instance		"naturalcover"
			minposition 0.3
			maxposition 0.7
			mincount 1
			maxcount 3
		}

		instanceonpath
		{
			instance		"tree_clump"
			minposition 0.2
			maxposition 0.8
			mincount 2
			maxcount 4
		}
	}

	mp_village_small
	{
		teamfilter		"blue"
		// team filter determines what color the instances will appear as
	on the automap and which team will spawn there
		filter			"ctf_blue"
		// any entity found in any instances defined within this section will
not appear if it has a filter and the filter doesn't match "ctf_blue"
```

```
                                        origin
                                        {
                                                node            "team_start"
                                        }

                                        instanceonpath
                                        {
                                                instance        "naturalcover"
                                                minposition 0.3
                                                maxposition 0.7
                                                mincount 1
                                                maxcount 2
                                        }

                                        instanceonpath
                                        {
                                                instance        "tree_clump"
                                                minposition 0.2
                                                maxposition 0.8
                                                mincount 2
                                                maxcount 4
                                        }

                                        instanceonpath
                                        {
                                                instance        "supplytent"
                                                minposition 0.2
                                                maxposition 0.4
                                        }


                                        instanceonpath
                                        {
                                                instance        "mp_random_defense"
                                                minposition 0.5
                                                maxposition 0.5
                                                pathalign       0.6
                                        }

                                        instanceonpath
                                        {
                                                instance        "ctf"
                                                teamfilter      "blue"
                                                filter          "ctf_blue"
                                                minposition 0.75
                                                maxposition 0.8
                                        }
                                }
                        } // end of instances
                } //end of standard
        } // end of file
```