

# Multiplayer Bots Manual

## Contents

- [1 Introduction](#)
  - [1.1 Preface](#)
  - [1.2 Bots Compatibility](#)
  - [1.3 Under Construction](#)
- [2 Console Commands](#)
  - [2.1 Add Bots](#)
  - [2.2 Kick Bots](#)
  - [2.3 Shutdown](#)
  - [2.4 Restart](#)
  - [2.5 Status](#)
  - [2.6 Skill Level](#)
  - [2.7 Team Avatars](#)
  - [2.8 Force Models](#)
  - [2.9 Bot Name Prefix](#)
  - [2.10 Script Loop Delay](#)
  - [2.11 Disallow Bots](#)
  - [2.12 Read-Only Variables](#)
- [3 Bots In Your Maps](#)
  - [3.1 Adding Bot Spawn Points](#)
  - [3.2 Creating Spawn Templates](#)
- [4 Pathnodes](#)
  - [4.1 Navigation Nodes](#)
  - [4.2 Cover Nodes](#)
  - [4.3 Debugging Nodes](#)
    - [4.3.1 Debug Tools](#)
    - [4.3.2 Erroneous Nodes](#)
    - [4.3.3 Run-Time Errors](#)
- [5 Bombs](#)
  - [5.1 Introduction](#)
  - [5.2 Insert a Bomb](#)
  - [5.3 Bots and Bombs](#)
    - [5.3.1 Plant Node](#)
    - [5.3.2 Defenders](#)
    - [5.3.3 Making Bots Look Smart](#)
    - [5.3.4 Script File](#)
- [6 Machineguns](#)
  - [6.1 Setting up the Turret](#)
    - [6.1.1 The Weapon](#)
    - [6.1.2 Navigation Info](#)
    - [6.1.3 Gun Trigger](#)
  - [6.2 Spotters](#)
  - [6.3 Turret Options](#)
    - [6.3.1 Gunner Info](#)
    - [6.3.2 Fire in Bursts](#)

- [6.3.3 Accuracy](#)
  - [6.3.4 Reloading](#)
- [7 Special Locations](#)
  - [7.1 Sniper Nodes](#)
  - [7.2 Camper Nodes](#)
- [8 Routes](#)
  - [8.1 Route Nodes](#)
  - [8.2 Routelists](#)
  - [8.3 Special Actions](#)
    - [8.3.1 Ladders](#)
- [9 Finishing Bots](#)
  - [9.1 Script File](#)
  - [9.2 Known Bug](#)
  - [9.3 Releasing Your Map](#)
    - [9.3.1 Future Bot Paks](#)
- [10 Advanced Bot Setup](#)
  - [10.1 Task Priorities](#)
  - [10.2 Weapon Priorities](#)
  - [10.3 Timelimit](#)
  - [10.4 Roundstart](#)
  - [10.5 Disabling Console Support](#)

# Chapter 1: Introduction

## 1.1 Preface

This is the manual for jv\_map's Multiplayer Bots version 1.0 for Medal of Honor: Allied Assault. Note that there is currently a newer bot version (1.1) available. The main concepts remain unaltered, but be aware some of the information in this document might not apply for the newer version. An upgraded manual is being assembled and will be online shortly. This manual describes how to enable bot support for your maps, how to use the (remote) console to control bots, how the script generally works and how to make your own modifications. There is also a FAQ available. If you think this manual isn't exactly clear about anything or if you still have questions, feel free to drop jv\_map a line.

## 1.2 Bots Compatibility

The multiplayer bots are developed for Medal of Honor: Allied Assault, not for Spearhead or other related games. Although some tests with Spearhead turned out to be succesful, this version remains unsupported. Bots can be added to any map as long as you have the .map file. This means, bots are available for any user map but not for stock maps.

## 1.3 Under Construction

Note that this manual isn't finished yet. Currently it's a basic guide that should get you going adding bots to your maps. A comprehensive manual is in the works but not finished yet. The latter will mainly include information on how to make your own additions and modifications to the bots. Moreover attention will be paid to the technical aspects of the script.

# Chapter 2: Console Commands

The bot script ships with a number of 'commands' that are available from the console. None of these are real commands; they're all cvars. This means, you can't just type the command but need to assign a value. For example, type:

```
jvbot_status 1
```

All console commands are handled from lib\_console.scr and are prefixed 'jvbot\_'. Only the server can issue these commands, either from the server console or via rcon. Example:

```
rcon jvbot_kickbot_team allies
```

Apart from these 'cvar commands', the bot script also uses three 'real' cvars, which are described below as well.

## 2.1 Add Bots

Possibly the most important command is to add bots. The basic way to do this is by typing:

```
jvbot_addbot_spawn 4
```

Where '4' is the number of bots to spawn. The newly spawned bots will automatically join the team with fewer members, so this is a good way to even teams. You can also set a team, which all bots will join. Set either

```
jvbot_addbot_team allies
```

To spawn Allied bots next time or set

```
jvbot_addbot_team axis
```

To spawn Axis bots. The team you've chosen is stored in a variable and will be read when you enter the 'jvbot\_addbot\_spawn' command. For example, to spawn 6 Axis bots, type the following:

```
jvbot_addbot_team axis  
jvbot_addbot_spawn 6
```

After you've typed in the spawn command, the team variable is reset.

Finally, you can spawn a bot with a set name. This only applies if you spawn just one bot at a time. You can set both a name and a team for the new bot. Let's say you want to spawn an Allied bot called 'Captain Ranger'. Type:

```
jvbot_addbot_name "Captain Ranger"  
jvbot_addbot_team allies  
jvbot_addbot_spawn 1
```

Note you'll have to use quotation marks for the bot name as it's more than one word. If you spawn a bot this way, the regular '[BoT]' prefix will not be used.

The model the bot uses and the weapon it chooses are merely random but will be consistent throughout the match.

## 2.2 Kick Bots

There are various ways of kicking bots. You can kick a particular bot, an entire bot team or just all bots. Particular bot kicking is generally for server administrators only, as you need to acquire the bot ID using the server console. This means you cannot use rcon.

If you want to kick a certain bot for bad behaviour, you need to find out his ID like stated above. To do so, use the 'jvbot\_status' command like explained below. For instance, if you found out you want to kick the bot with ID 4, type:

```
jvbot_kickbot_id 4
```

Easier ways to kick bots are using the team kick command or the kick all command. For example, to kick all Allied bots, type:

```
jvbot_kickbot_team allies
```

All Allied bots will leave the battle within seconds.

To kick all bots, type:

```
jvbot_kickall 1
```

The '1' just tells the script to execute this command. The kickall command should only be used to kick all bots and spawn a couple of new bots a few moments later. If you want to permanently clear your server of bots, you're better off using the shutdown command as described below.

## 2.3 Shutdown

Another revolutionary feature included in the bot script is that it allows a full shutdown. All bots will be removed and all threads except for the console thread will be exited. This allows you to free up the game in case anything weird has happened and continue to play without bots. The server will gain a little bit of CPU time. You can always restart the bot script later.

To shut down the bot script, type:

```
jvbot_shutdown 1
```

A full shutdown takes about 3 seconds.

## 2.4 Restart

After a shutdown, you may use the restart command to allow bots on your server again. All bots initially inserted by the map's script will be re-added. For a restart, type:

```
jvbot_restart 1
```

## 2.5 Status

Furthermore, a status command is available, which will print the current bot status to the server console. So, it doesn't have much use typing this command from a remote console (rcon). The status command gives you info about all bots' ID, team, state, name, task and how they were spawned. For example, the command

```
jvbot_status 1
```

Could print the following:

```
# team state name (task) [spawner]
-----
1 allies active [BoT]Dude (attackbomb) [auto]
2 axis  dead [BoT]Heinrich (none) [auto]
12 axis  prepare [BoT]Dangerous (none) [manual]
13 none  entering [BoT]Lt. King (none) [manual]
-----
```

Looking at the first bot, this provides the following info: '[BoT]Dude' has bot ID 1, is a member of the Allied team, is currently attempting to plant a bomb ('attackbomb') and was spawned from the level script ('[auto]'). A bot's 'state' can be any of the following:

- Entering: The bot has joined the game but not yet chosen a team.
- Prepare: A bot is already spawned in the game but in 'ghost' mode, it's waiting for a task before being activated.
- Active: This bot is currently active and performing tasks.
- Dead: The bot died and is waiting to be respawned.

If the 'spawner' of a bot is set to '[manual]', this bot was spawned from the console. The spawner variable allows level designers to keep track of their bots when spawning in groups.

## 2.6 Skill Level

The bot script allows four different skill levels for the bots, ranging from '1' (very easy) to '4' (hard). This affects bot aiming accuracy and speed as well as health level. Bot skill defaults to '2'.

To set a different bot skill level, for instance level 3, type:

```
jvbot_skill 3
```

## 2.7 Team Avatars

For easy team identification, the bot script automatically spawns a team avatar above the heads of all bots. Unlike the headicons of regular players, all players will be able to see these avatars. This seems like an unresolvable problem, which can be annoying in a sniper map. Therefore you can disable all team avatars for bots using one simple command:

```
jvbot_showavatar 0
```

Later, you can use

```
jvbot_showavatar 1
```

To enable bot head icons again.

As the 'showavatar' cvar is not cleared at start-up, map designers can decide to set this variable in their map's script. To do this, just add:

```
setcvar "jvbot_showavatar" "0"
```

In the map's script below 'level waittill spawn' just before the line that starts the bot script.

## 2.8 Force Models

In order to improve performance, you can force a server to use the same model for all bots from one team. This reduces the number of textures used. To force all bot models to one model, type:

```
jvbot_forcemodels 1
```

Later, you can type

```
jvbot_forcemodels 0
```

To disable model-forcing again. The model chosen is the first element of the 'alliesspawnpreset' and 'axisspawnpreset' array. You can only change the model used by changing these models in the map file.

All bot model forcing (and 'unforcing') is applied the next time a bot respawns.

## 2.9 Bot Name Prefix

By default, the names of all bots spawned are prefixed with a '[BoT]' tag to designate them as bots. However, you can change this prefix at any time, although the changes only affect bots that still have to enter. To change the prefix to e.g. '-clan-', type:

```
jvbot_prefix "-clan-"
```

To omit prefixes altogether, type:

```
jvbot_prefix "NULL"
```

You may also set (or clear) the default bot prefix in the map's script. Use:

```
setcvar "jvbot_prefix" "-clan-"
```

Or, to clear all prefixes:

```
setcvar "jvbot_prefix" "NULL"
```

## 2.10 Script Loop Delay

Because bot scripts have an impact on the server CPU usage, the script allows you to control the number of calculations per second. You can set a so-called 'loop delay', which is the time in seconds the script waits before processing a bot again. If you set this number higher, the script will perform less calculations per second and the server performance should improve. However, bots will be less responsive and tend to look stupid if you set the value very high. The lowest value is 0, which means all bots will be processed every frame. This has an enormous impact on server CPU usage.

You can set a loop delay at any time, using the 'jvbot\_loopdelay' cvar. Default is 0.4. If you want to set it to 0.8, type the following:

```
jvbot_loopdelay 0.8
```

You can set or change the loopdelay value at any time in the level script using setcvar:

```
setcvar "jvbot_loopdelay" "0.8"
```

## 2.11 Disallow Bots

In case you don't want to spawn any bots on your server, you can simply set a cvar to prevent them from entering the battle. Add the following line to the config file of the server:

```
sets "jvbot_allowed" "0"
```

If you change your mind while in game and want to spawn bots nonetheless, you just have to start the bot script using the 'jvbot\_restart' command as described above.



## 2.12 Read-Only Variables

The bot script uses two cvars that are meant to be read only. One is 'jvbot\_running', which is '1' if the bot script is currently running and '0' otherwise. The other variable is 'jvbot\_version', which is no more than the version number of the bot script.

# Chapter 3: Bots In Your Maps

This chapter describes some basic modifications you have to make to your map to allow bots.

## 3.1 Adding Bot Spawn Points

You can spawn bots at any entity that has an origin. In general, use the script\_origin entity. Do not spawn bots at player spawn positions, as you'll risk players getting stuck in bots. When you've created the script\_origins, targetname them 'alliesspawn' for an Allied bot spawn position or 'axisspawn' for an Axis bot spawn position. The bot script will randomly choose between spawn points.

## 3.2 Creating Spawn Templates

Apart from spawn positions, the bot script also needs to have a list of models that can be used for bots. The script allows you to place these anywhere in your map. The bot script ships with a load of available models. You can easily add all common bot models using the prefab supplied in the bot SDK, which is available from the Freebrief site.

If you want to add more bot models, just place the desired ones (in the ai -> multiplayer popup menu) in your map and assign a targetname of either 'alliesspawnpreset' for an Allied bot template or 'axisspawnpreset' for an Axis model. You can add any number of spawn templates you like. If you add multiple entities with the same model, this particular model will be used more. Thus, you can create some kind of 'model weighting'.

# Chapter 4: Pathnodes

## 4.1 Navigation Nodes



Example pathnode placement

For bot navigation purposes, your map literally has to be scattered with info\_pathnode entities. All info\_pathnodes in your map are combined into a navigation file (.pth), which is generated when you first load your map. You should include the .pth file when releasing your map. This is also the reason why it seems impossible to add bots to stock maps: these maps don't have pathnodes.

Since the game has to create a navigation file from your pathnodes, their placement must be careful and has to meet a number of requirements. First, pathnodes can't be more than 320 units apart. Also, there should be a free 'tunnel' between pathnodes of at least 32 units wide and 116 units high. This means you cannot add pathnodes in small walkways or ventilation shafts; bots won't be able to go there. Finally, bots can't climb an incline steeper than 45 degrees. If you need your bots to, try to add a small 'common/monster' ('botclip') brush to smooth the slope. You're only done when there are pathnodes all over your map, so bots can reach every area when following human players.

You can mark a node with spawnflag 'dont\_link'. This allows you to reduce the number of paths and thereby increase performance. If you want to remove a path between two pathnodes, add a dont\_link node in between. However, this functionality serves little use for multiplayer bots.

## 4.2 Cover Nodes

To make the bots look smarter, it is possible to mark certain pathnodes as 'cover' or 'concealment'. These tags can be set as spawnflags for any pathnode. Cover means that a bot will consider this location a good place to hide (for example to reload) unless its enemy can see the cover location. A concealment node is always considered good cover, so only use it for bushes or other 360 degrees cover objects.

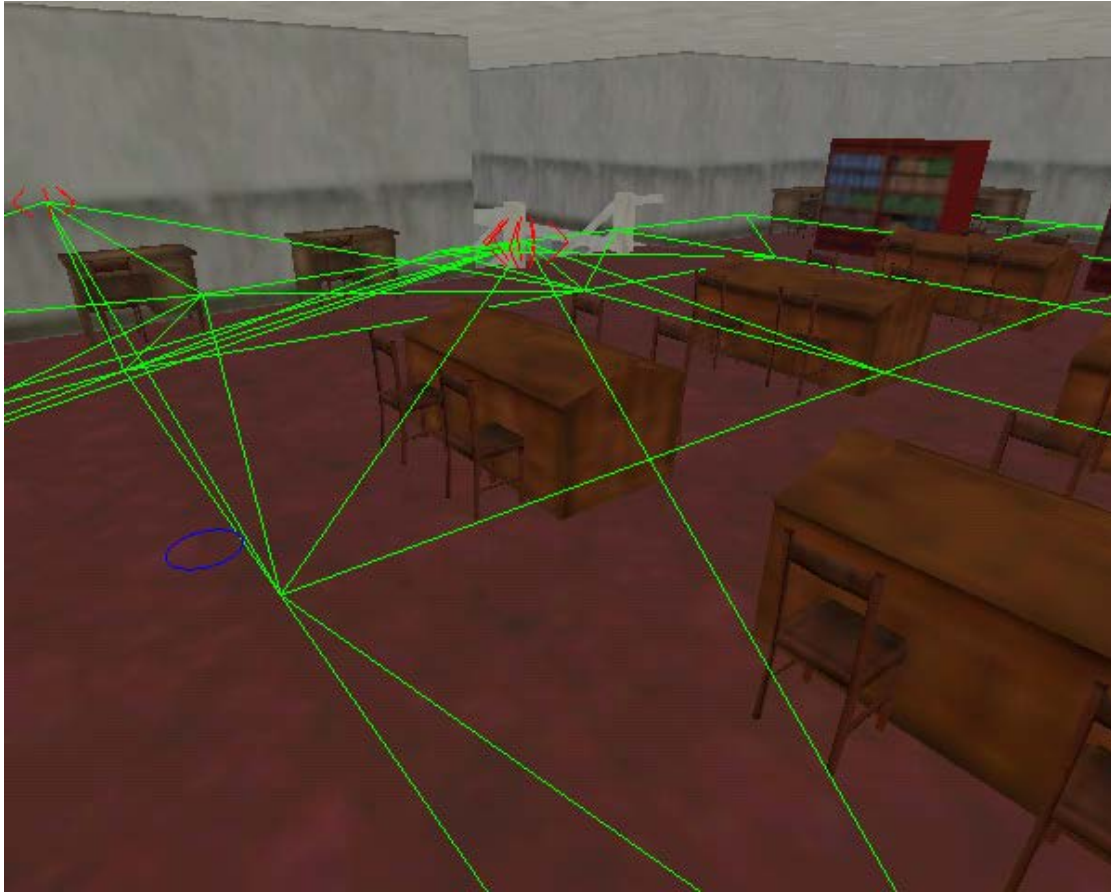
Apart from this, you can also set 'duck', 'corner\_left', 'corner\_right' and 'crate'. Duck means that a bot will crouch when moving to this node while under attack. This usually looks stupid. Both corner flags are used by the game to determine good locations to shoot around a corner. However these spawnflags cause the game to use the single player shoot-around-the-corner animations, which makes the bots look less 'human' in multiplayer games. Crate is even worse. It's used to allow AI characters to fire over the top of a crate, but often causes erratic attacks by bots. So best don't use any of these special spawnflags.

## 4.3 Debugging Nodes

### 4.3.1 Debug Tools

Regardless of your skill, you are bound to create a couple of flaws in your pathnode layout. Fortunately, MoH:AA ships with a very useful debugging tool. In the console, type 'togglmenu leveldesign', which will show a small level designer's menu. Now click the 'routes' button. You will see a number of green lines. These lines indicate all paths a bot can follow to reach its destination. You'll probably also notice a couple of symbols, which are described below.

### 4.3.2 Erroneous Nodes



Debugging pathnodes

You might happen to see red crosses on some locations. These indicate unreachable pathnodes. Such a node is either too far away from other pathnodes or too close to complicated architecture. It's important to fix all red crosses before releasing your map, or the bots will have a hard time navigating around. Also, you might find a couple of red arrows. These arrows indicate 'one way' paths, which means that a bot can only follow these paths in the direction the arrows point to. This usually happens when creating paths on a steep slope or near architecture. Finally, you may notice small blue circles. These represent nodes marked with the 'dont\_link' flag.

### 4.3.3 Run-Time Errors

When play-testing your map, you might notice some 'Couldn't find end node' or 'Unreachable path' errors. This means a bot has had problems to reach his destination. It's very hard to prevent all of these errors, but if you notice a lot, it's a good idea to take another close look at the pathnodes and improve their layout whenever possible. It's some work but it will result in smarter bots in the future.

# Chapter 5: Bombs

## 5.1 Introduction

You will virtually always need bombs in your objective maps. Therefore, a special bomb script was written which allows both players and bots to plant and defuse bombs. Moreover, it allows you to assign different bombs to different teams and to choose a custom explosive model and countdown time. How to do this is discussed further down.

This chapter focuses on how to set up bombs so bots can find, plant and protect them. Remember to include the necessary script commands as described below.

## 5.2 Insert a Bomb

To add a bomb, you first need to place a script\_model anywhere in your map where you want the explosive pulse model to show up. Assign a model by typing key 'model' / value 'items/pulse\_explosive.tik'. This model is used as the pulse overlay. When you've done this, give your bomb a targetname so your script can recognize it. Then, set key '\$plantteam' to the team that has to plant the bomb, so either 'allies' or 'axis'. If you don't set a \$plantteam, this will default to the team stored in 'level.planting\_team', or to 'allies' if this value isn't set either.

Next, add a trigger\_use which can be used by players trying to plant a bomb. Make the trigger\_use slightly larger than the bomb, and give it a targetname. Then add the following key / value set to your bomb model: key 'trigger\_name' / value [targetname of trigger to use]. Players can now plant a bomb here.

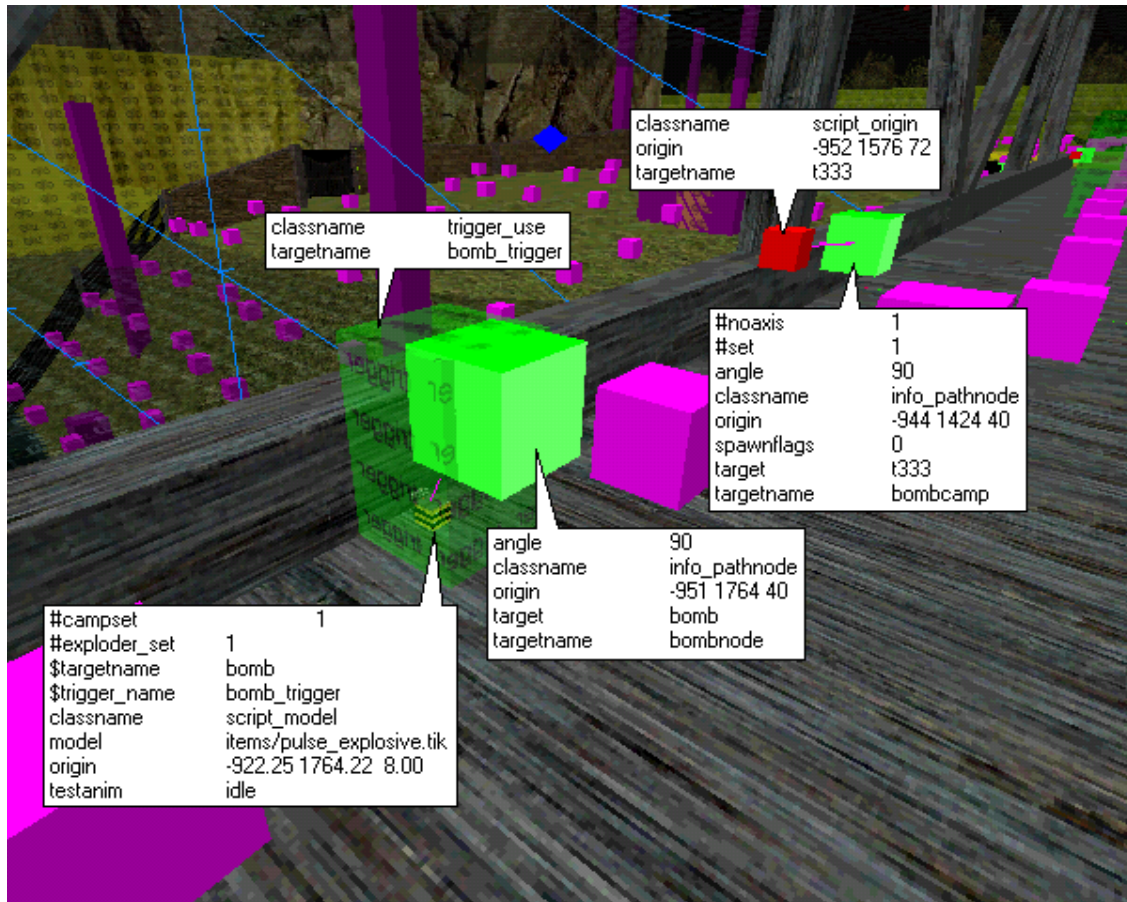
## 5.3 Bots and Bombs

### 5.3.1 Plant Node

You probably want to enable bots to plant bombs too. Therefore, you need to tell the bots where your bomb is. Do so by adding an info\_pathnode in front of your bomb. Assign a targetname of 'bombnode'. To connect the pathnode and the bomb, first select the bombnode, then the bomb and press CTRL + K.



### 5.3.2 Defenders



Example bomb placement

Also, you'll want your bots to defend the bomb wherever possible. This works with so-called 'bombcamp' nodes; info\_pathnodes with targetname 'bombcamp'. Add a number of pathnodes around your bomb at all spots that seem good places to defend the bomb. As a rule of thumb, add at least eight pathnodes per bomb. All these info\_pathnodes should have targetname 'bombcamp'. Remember the more bombcamp nodes you add, the more varied bot play will be.

You need to link the bombcamp nodes to the bomb. This works using a #set key. All bombcamp nodes should have an identical value for this. So, for example use key '#set' / value '1' for bombcamp nodes near your first bomb. Then assign key '#campset' / value '1' for the corresponding bomb entity.

If all went well, bots are now able to plant and defend your bomb.

### 5.3.3 Making Bots Look Smart

If you want bots to aim at a certain location when defending the bomb - for example a corridor that allows attackers access to the bomb, create script\_origins at these spots. Then select one (or more) of your bombcamp nodes and target it to the script\_origin using the CTRL + K method as described above (first select the pathnode, then the script\_origin). A bot will now aim at the given origin when defending at the corresponding pathnode.

Just in case you don't want Allied bots to use a certain camp node, you can set key '#noallies' / value '1' to stop them. On the other hand, you can set key '#noaxis' / value '1' to prevent Axis bots from defending the bomb at that particular spot. These key / value sets should be applied to the info\_pathnode, not to the bomb.

### 5.3.4 Script File

There's one last thing you need to do to get the bombs working. A so-called 'bomb thinker' has to be started for each bomb. Let's say you're creating a bomb with targetname 'mybomb', then add the following line to your map's script file just below 'level waittill spawn':

```
$mybomb thread global/jv_obj_dm.scr::bomb_thinker
```

You need to add this line for any bomb in your map.

## Chapter 6: Machineguns

The multiplayer bot script allows easy implementation of machinegun turrets in any map. The machinegun script attacks random threats within a set range. It checks for targets at a full 360 degrees of freedom. If a threat can't be engaged with the turret, the gunner will undertake action with his handarm or the supporting spotter bot will be deployed to deal with the target.

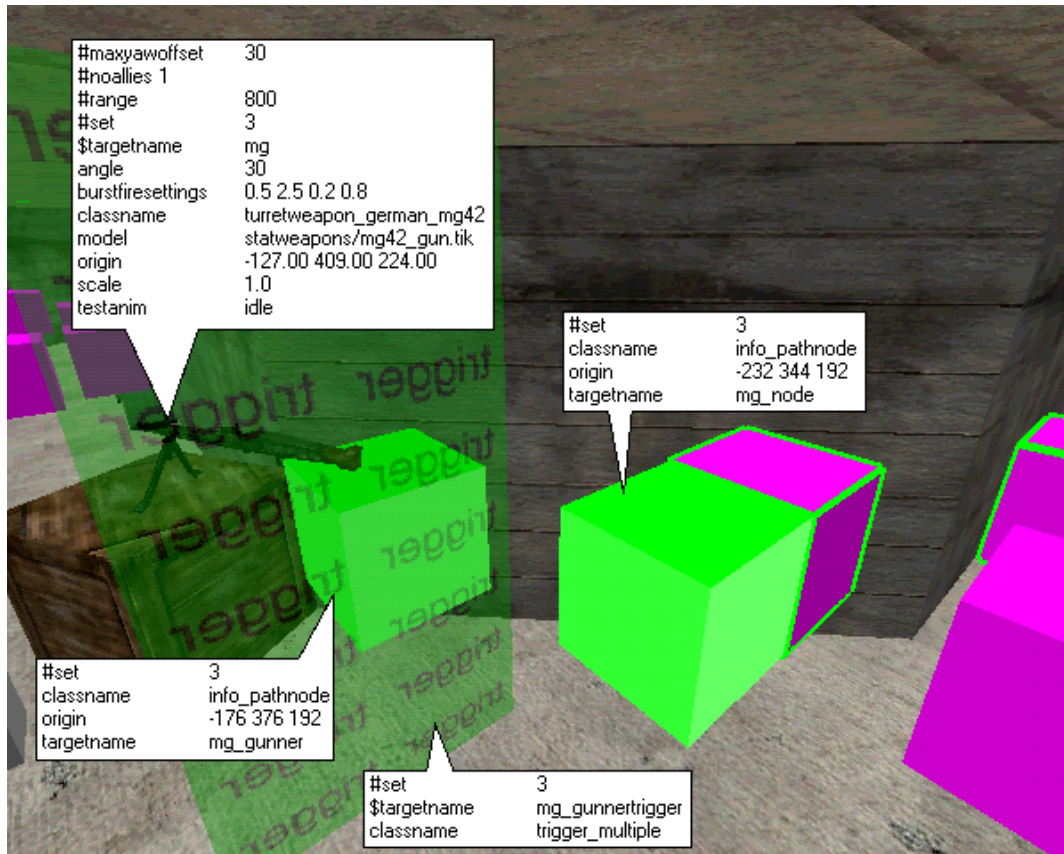
### 6.1 Setting up the Turret

#### 6.1.1 The Weapon

It's not particularly difficult to set up a turret for bot use. There are just a few things you need to take care of.

First, insert a turret gun with targetname 'mg' and set a '#maxyawoffset' key. Don't set it without the '#'; it is of maior importance for the script. Next, it's a good idea to set a '#range' key which determines the maximum target search range in game units. Default is 4000 units, but setting an appropriate range is very important to save CPU usage. Finally add a '#set' key, this should be a unique value for each gun. You'll need it later on.

### 6.1.2 Navigation Info



#### Example machinegun setup

Now you need to add some navigation info so the bots can actually reach your gun. You need at least two pathnodes for a gun. One is an approach node, where bots will move to when planning to man a gun. If a gun is already in use, the bot will walk away and perform an other task. So create this 'approach node' at a small distance from your gun, about 64 units. It should be an info\_pathnode with targetname 'mg\_node' and a '#set' value identical to the #set of the corresponding gun.

One more pathnode has to be made. This is the node where the actual gunner will be standing when firing the weapon, so place it just behind the gun. Targetname should be 'mg\_gunner'; #set identical to the gun's #set.

### 6.1.3 Gun Trigger

Finally, create a trigger\_multiple around your gun. This trigger should envelop the area just behind the gun, where the gunner is standing. This trigger is used to determine whether a player is using a gun. Targetname is supposed to be 'mg\_gunnertrigger'. Also set the correct #set key / value.

Your gun now is good to go.



## 6.2 Spotters

The bot script also allows one spotter bot per machinegun. This bot will notify the gunner of new targets and immediately take over the gun when the gunner is shot. This will make your gun more effective and fearsome.

To add a spotter to your turret, add another info\_pathnode. Place it slightly to the left of the gunner node, about 32 units. Assign a targetname of 'mg\_spotter' and a #set value which equals the #set value of your gun.

## 6.3 Turret Options

A number of variables can be set to modify the gun turret to your needs. Here's a complete list. All key / value pairs apply to the turretgun entity.

### 6.3.1 Gunner Info

First, you can set a '#noallies' or a '#noaxis' key. Value should just be '1'. This key will disallow either Allied or Axis bots to use this gun. You can also add a '#weight' key, which determines the tendency of bots to use this particular gun. Weights default at 1. If you set a higher value, bots are more likely to use this gun.

### 6.3.2 Fire in Bursts

Moreover the gun might look more realistic if you set a 'burstfiresettings' key. Actually this key hasn't got anything to do with the bot script; it's implemented in the game code. This key allows you to specify a set time of firing and a delay between firing bursts. You need to specify four values, mintime, maxtime, mindelay and maxdelay. These values control the duration of firing bursts as well as the time a delay between bursts takes.

### 6.3.3 Accuracy

Next, the bot script allows you to set an '#accuracy' key, which should be a value between 0 and 100. By default this value is set at 95. The accuracy is automatically modified when the server changes bot skill level. At default skill level, an accuracy of 0 means a 45 degree cone of bullet spread, which is excessive.

### 6.3.4 Reloading

By default, bots using the machinegun will reload the gun every once in a while. To stop this behaviour, type key '#noreload' / value '1'.

# Chapter 7: Special Locations

## 7.1 Sniper Nodes

ju\_map's bot script also allows the placement of sniper nodes anywhere in your map. Bots will stand in place at these locations and aim at a given target. To add a sniper node, just create an info\_pathnode somewhere with targetname 'sniper '. Then add a (about 256 units off) script\_origin where you want a sniper bot to aim at. First select the sniper node, then the script\_origin and press CTRL + K. Your sniper node is ready.

However, you may want to modify the sniper node to your demands. First, you can prevent bots from initially crouching down at a sniper node by setting a key of '#nocrouch' with value '1'. Also, you may set the regular '#noallies' and '#noaxis' keys. Finally you can set a '#weight' key to increase the importance of a particular node. Weights default to 1.

## 7.2 Camper Nodes

Apart from sniper nodes, also camper nodes are available. They work very similar. In general, use sniper nodes for long range hiding places, and camper nodes for ambush positions near for example small corridors. Bots with short range weapons such as shotguns or sub-machine guns have a tendency to use camper nodes.

Create camper nodes like sniper nodes. Add an info\_pathnode with targetname 'camper' targeted to a script\_origin to aim at. You may also assign the '#noallies', '#noaxis' and '#weight' keys. '#nocrouch' is not available, as bots won't attempt a crouch down at camper nodes.

# Chapter 8: Routes

When running, bots always take the shortest route to their destination. This tends to make their behaviour very predictable. Therefore a route system has been created, so you can setup surprise attacks and increase the variety of the game. Also, the route system allows bots to use special ways of reaching their destination, for example by climbing a ladder.

## 8.1 Route Nodes

Routes are created by a number of info\_pathnodes. You can create a route to anything you like, for example bombs or machineguns. Just add info\_pathnodes along the way you want bots to walk on. You don't need to add many, just a couple defining the basic layout of the route.

When you've created the pathnodes, it's time to designate them as route nodes. To do so, select them all and targetname them 'route'. Moreover, all nodes forming one route should have an identical '\$name' key, which is used as a name for the route. Set it to whatever you can think of.

You'll have to define the direction of movement. This works with an '#id' key. Add key '#id' and value '1' for the node nearest to the destination of the route. Enumerate up to the last node. So if you're creating a route with 4 nodes to a bomb, the node nearest to the bomb should have #id 1 and the last node #id 4.

Now your route is ready to be used, provided you tell the bots when they can use this route. You'll have to define the destination(s) of the route. This works with a '\$route' key, which is set on the entity the route leads to. So, if the destination of your route is a \$bombnode entity, add the \$route key to this bombnode entity. The value should be the '\$name' key of your route nodes. You can add any number of destinations to one route, so you could for example also add the same \$route key to all \$bombcamp nodes. Read the next section ('Routelists') on how to setup multiple routes to one entity.

One last note about the first node (lowest #id) of your route. A bot will never walk to this node. It's just used to determine the direction of the route. So, when you've designed a route, it could be a good idea to add another 'dummy' node with a higher #id. This way the original first node of the route will still be used.

## 8.2 Routelists

As the system with the '\$route' key only allows one route per entity, there's another key you can use. This requires some scripting though, but will make your bots look smarter. It works with a so-called 'routelist', which is literally a list of all routes that lead to an entity.

If you're planning to use a routelist, you have to set it up in your level script. You have to do so before enabling the bots. First choose a name for your routelist. This must be a unique name for every routelist, but you can use the same routelist for multiple destination entities. The variable name for your routelist will be 'level.routelist[name]'. Now all you have to do is add the right route names to the array. For example, let's say you're creating a routelist with name 'routes\_to\_bomb', which contains the routes with \$name keys 'north', 'lower east' and 'south'. The line for the routelist would be the following:

```
level.routelist[routes_to_bomb] = "north"::"lower east"::"south"
```

Or you could type:

```
level.routelist[routes_to_bomb][1] = "north"  
level.routelist[routes_to_bomb][2] = "lower east"  
level.routelist[routes_to_bomb][3] = "south"
```

Finally, add key '\$routelist' with as value the name of your routelist to every entity you want to apply this routelist to. For example, if the 'routes\_to\_bomb' routelist from the previous example would contain all routes to a bombnode entity, add key '\$routelist' and value 'routes\_to\_bomb'.

## 8.3 Special Actions

The route system allows special bot actions at any node of the route. These actions can make your bots look smarter and more realistic. The essence of the system is that no regular navigation info is required, since it's all stored in the routes and routelists. This allows bots to navigate outside the scope of the pathnode navigation system that's used by the game AI.

To setup a special action, you need to add an '\$action' key at a route node. Any bot that reaches this node when following the route it belongs to, will perform the action defined. All special actions should be in a separate script file, which is named 'special\_(action).scr' and located within the bots folder. The value of the \$action key should be the suffix of the action script. For example, to use the script 'special\_crouch.scr' (a possible application of the route system), the \$action key should be 'crouch'. The ladder script is currently the only special action script available.

Moreover, one additional parameter is allowed which is set by using the '#actionset' key. Generally, this key should just be a unique number to discern which node triggered the action. For instance, this could determine which ladder a bot should use. You may use the same #actionset values for different actions.

As the first node of a route (highest #id) is always skipped by bots, it has no use to add an \$action key to such a node. If you need to do so, add another node just in front with a higher #id.

### 8.3.1 Ladders

This section describes how to set up bot-climbable ladders. First, you need to define a route with an action node. Place one node in front of the ladder. Add key \$action with value 'ladder'. Also set an '#actionset' key, which is a unique number per ladder. Assuming this is your first ladder, this value should be 1.

The setup of the actual ladder is slightly more complicated. You need to add two script\_origins, one at the bottom of your ladder and one at the top. Place them about 32 units off the ladder surface. The bottom script\_origin should be placed exactly at the floor underneath the ladder. The top entity should be at the level of the higher floor. Targetname the bottom entity 'ladderbottom' and the top origin 'laddertop'. Give both entities a #set key which is equal to the #actionset value of the corresponding route node, for example 1.

Finally you need to define the angle of a bot climbing the ladder. This should be done using the 'angles' key. To find the appropriate angles, just apply a regular angle first. Make both origins point at the ladder. Then, add an angles key to both entities which should consist of three numbers: pitch, yaw and roll. As you'll generally only want to modify the yaw the bots use, the first and last number should just be a 0. So, add a value of '0 (angle) 0', for example '0 90 0' for an angle of 90. Then, remove the angle key. The reason for this expedient method is that the script isn't able to read the angle key from the map.

Your ladder is now ready to be used. Bots will climb or descend it whenever following the route the ladder is part of. If you find the bots behaviour towards ladders doesn't look very well, you can often improve this by nudging the top and bottom nodes a bit. Also make sure

you test the bots by shooting them while they're climbing a ladder. If they fly upwards after they died, you need to increase the distance of bottomnode and topnode to the ladder surface.

## Chapter 9: Finishing Bots

### 9.1 Script File

To get working bots in your map, you need to add a line to your map's script file. Thanks to the easy interface of the bot script, this isn't a difficult job. Add the following line below level waittill spawn. If you have any bombs in your map, you need to add this line below all 'bomb\_thinker' lines.

```
waitthread global/jv_bots/jv_mp_ai.scr::enable
```

You might also want to spawn a couple of bots when your map starts. Otherwise, you can only add bots by console. To spawn a set number of bots when your map loads, type:

```
level.alliesbots = (number of Allied bots)
level.axisbots = (number of Axis bots)
```

Paste this code just above the enable line from the previous example.

In the end, a part of your script could look like the following example:

```
level waittill spawn
$bomb thread global/jv_obj_dm::bomb_thinker
level.alliesbots = 5
level.axisbots = 5
waitthread global/jv_bots/jv_mp_ai.scr::enable
```

### 9.2 Known Bug

There's (at least) one major bug in the bots script. If you don't set a farplane (fog) in your worldspawn, bots will be completely blind, so they won't attack. You need to define a farplane distance, even if it's very large. This bug will be fixed in the next bot release though.

### 9.3 Releasing Your Map

Finally it's necessary to include a couple of files when releasing your map. All files needed are in a 'jv\_botx.pk3' file, where 'x' is a version number. You can always get the latest bot version at the Freebrief site.

Include the bot pk3 file in the zip file that contains your map's .pk3 file. In the readme.txt file accompanying your map, state that the user has to extract both .pk3 files to his 'main' folder. Do not rename the bot .pk3 file. If you've altered the scripts inside, distribute these using a completely different filename. This will prevent version conflicts in the future.

### 9.3.1 Future Bot Paks

New versions of the bot pk3 are likely to be developed. The developer's aim is to make all versions compatible. To ensure this, you can test any newer versions of the bots pk3 on your map before these are going public. This way you can make sure newer bot versions don't harm bot play in your map. Nonetheless, the author of the bot scripts can never be held responsible for any damage they do. If you want to be notified about newer bot versions, just send jv\_map an e-mail stating this. The e-mail address can be found at the Contact Page of the Freebrief site.

## Chapter 10: Advanced Bot Setup

You might want to modify bot behaviour to map specific needs. You can change the priorities for certain tasks and weapons. This chapter is meant to help you do so. Moreover, the bot `timelimit` and `roundstart` events are discussed.

### 10.1 Task Priorities

The easiest way of modifying bot behaviour is by changing their priorities for certain tasks. If you for example think bots are trying a direct bomb assault too much - instead of sniping from a distance - you could try to either raise the sniper priority or drop the priority of the bomb planting behaviour. You can do so in your level script.

The following tasks are by default implemented in the bot script:

- `attackbomb`: Plant and protect planted bombs (default priority: 1).
- `defendbomb`: Defuse and defend bombs (default priority: 1).
- `follow`: Follow players (default priority: 1.5)
- `machinegun`: Use a machinegun turret (default priority: 0.5)
- `sniper`: Stand in place and engage long-range targets (default priority: 0.1)
- `camp`: Camp at a \$camper node to ambush the enemy (default priority: 0.1)
- `health`: Quick task to pick-up a medkit (default priority: 10)

To change the priority of a certain task, you must change the priority value as stored in the `'level.jvbot_tasks_priority'` array. For example, to assign a task priority of 2 for the sniper task, type:

```
level.jvbot_tasks_priority[sniper] = 2
```

You can set this value anywhere in your map script. Preferably set it before calling the bot script.

## 10.2 Weapon Priorities

You may also change the bot tendency to use certain weapons. This works with the 'level.jvbot\_weapon\_priority' array. For example, to set a weapon priority of 50 for the M1 Garand, type:

```
level.jvbot_weapon_priority["m1 garand"] = 50
```

You can change the priority of all weapons at any time. Preferably set your custom priorities before calling the bot script. Below is a list of all available weapons and their priorities. Note that a priority of 0 means that this weapon will never be used.

```
level.jvbot_weapon_priority["m1 garand"] = 10
level.jvbot_weapon_priority["springfield '03 sniper"] = 20
level.jvbot_weapon_priority["thompson"] = 40
level.jvbot_weapon_priority["bar"] = 25
level.jvbot_weapon_priority["shotgun"] = 5
level.jvbot_weapon_priority["high standard"] = 0
level.jvbot_weapon_priority["colt 45"] = 0
level.jvbot_weapon_priority["bazooka"] = 0
level.jvbot_weapon_priority["mauser kar 98k"] = 5
level.jvbot_weapon_priority["mauser kar 98d sniper"] = 20
level.jvbot_weapon_priority["mp40"] = 30
level.jvbot_weapon_priority["stg44"] = 40
level.jvbot_weapon_priority["walter p38"] = 0
level.jvbot_weapon_priority["panzerschrek"] = 0
```

## 10.3 Timelimit

For objective matches, the game normally waits till both teams consist of at least one player. However, the game doesn't recognize bots as team members. That's why a special timelimit script was written ('lib\_timelimit.scr'). This script starts another timer below the regular timer, so you'll see two timers when playing the map. The game ends whenever any of them hits 0; usually the lower ellapses first. You can at any time change the timelimit using the 'timelimit' command. If you want to disable the bot timelimit, type:

```
timelimit -1
```

In case you for some reason don't want to use the timelimit script on your map, just add the following line to your script above the bot 'enable' line:

```
level.jvbot_notimehandler = 1
```

## 10.4 Roundstart

Just a quick note to scripters. You'll usually want to prevent one team from winning as long as the other team doesn't have any members. Therefore the 'level.roundstart' variable was introduced. It's set to 1 as soon as both teams have at least one player or bot. Example of a basic 'wait till roundstart' script:

```
while !(level.roundstart)
    waitframe
```

## 10.5 Disabling Console Support

It's possible to disable all console based bot commands if you want to. There doesn't seem any real use for this, but if you want to disable console support for your map, just type:

```
level.jvbot_noconsole = 1
```

In your map script before calling the bot script.