Warzone2100 JavaScript Scripting API

1 Introduction

Warzone2100 contains a scripting language for implementing AIs, campaigns and some of the game rules. It uses JavaScript, so you should become familiar with this language before proceeding with this document. A number of very good guides to JavaScript exist on the Internet.

The following hard-coded files exist for game rules that use this API:

multiplay/skirmish/rules.js Basic game rules - base setup, starting research, winning and losing.

multiplay/script/scavfact.js Scavenger AI. This script is active if scavengers are.

For ordinary AI scripts, these are controlled using '.ai' files that are present in the 'multiplayer/skirmish' directory. Here is an example of an '.ai' file that defines an AI implemented using this API:

```
[AI]
name = "Semperfi JS"
js = semperfi.js
```

It references a '.js' JavaScript file that needs to be in the same directory as the '.ai' file. The code in this file is accessed through specially named functions called 'events'. These are defined below. An event is expected to carry out some computation then return immediately. The game is on hold while an event is processed, so do not do too many things at once, or else the player will experience stuttering.

All global variables are saved when the game is saved. However, do not try to keep JavaScript objects that are returned from API functions defined

here around in the global scope. They are not meant to be used this way, and bad things may happen. If you need to keep static arrays around, it is better to keep them locally defined to a function, as they will then not be saved and loaded.

One error that it is easy to make upon initially learning JavaScript and using this API, is to try to use the 'for (... in ...)' construct to iterate over an array of objects. This does not work! Instead, use code like the following:

```
var droidlist = enumDroid(me, DROID_CONSTRUCT);
for (var i = 0; i < droidlist.length; i++)
{
        var droid = droidlist[i];
        ...
}
```

The above code gets a list of all your construction droids, and iterates over them one by one.

The droid object that you receive here has multiple properties that can be accessed to learn more about it. These propertie are read-only, and cannot be changed. In fact, objects that you get are just a copies of game state, and do not give any access to changing the game state itself.

Any value written in ALL_CAPS_WITH_UNDERSCORES are enums, special read-only constants defined by the game.

2 Common Objects

Some objects are described under the functions creating them. The following objects are produced by multiple functions and widely used throughout, so it is better to learn about them first.

2.1 Research

Describes a research item. The following properties are defined:

power Number of power points needed for starting the research.

points Number of resarch points needed to complete the research.

started A boolean saying whether or not this research has been started by current player or any of its allies.

done A boolean saying whether or not this research has been completed.

name A string containing the canonical name of the research.

2.2 Structure

Describes a structure (building). It inherits all the properties of the base object (see below). In addition, the following properties are defined:

status The completeness status of the structure. It will be one of BEING_-BUILT, BUILT and BEING_DEMOLISHED.

type The type will always be STRUCTURE.

stattype The stattype defines the type of structure. It will be one of HQ, FACTORY, POWER_GEN, RESOURCE_EXTRACTOR, DEFENSE, WALL, RESEARCH_LAB, REPAIR_FACILITY, CYBORG_FACTORY, VTOL_FACTORY, REARM_PAD, SAT_UPLINK, GATE and COMMAND_CONTROL.

2.3 Feature

Describes a feature (game object not owned by any player). It inherits all the properties of the base object (see below). In addition, the following properties are defined:

type It will always be FEATURE.

2.4 Droid

Describes a droid. It inherits all the properties of the base object (see below). In addition, the following properties are defined:

type It will always be DROID.

order The current order of the droid. This is its plan. The following orders are defined:

DORDER_ATTACK Order a droid to attack something.

DORDER_MOVE Order a droid to move somewhere.

DORDER_SCOUT Order a droid to move somewhere and stop to attack anything on the way.

DORDER_BUILD Order a droid to build something.

DORDER_HELPBUILD Order a droid to help build something.

DORDER_LINEBUILD Order a droid to build something repeatedly in a line.

DORDER_REPAIR Order a droid to repair something.

DORDER_RETREAT Order a droid to retreat back to HQ.

DORDER_PATROL Order a droid to patrol.

DORDER_DEMOLISH Order a droid to demolish something.

action The current action of the droid. This is how it intends to carry out its plan. The C++ code may change the action frequently as it tries to carry out its order. You never want to set the action directly, but it may be interesting to look at what it currently is.

droidType The droid's type.

group The group this droid is member of. This is a numerical ID. If not a member of any group, this value is not set. Always check if set before use.

2.5 Base Object

Describes a basic object. It will always be a droid, structure or feature, but sometimes the difference does not matter, and you can treat any of them simply as a basic object. The following properties are defined:

type It will be one of DROID, STRUCTURE or FEATURE.

id The unique ID of this object.

x X position of the object in tiles.

y Y position of the object in tiles.

z Z (height) position of the object in tiles.

player The player owning this object.

selected A boolean saying whether 'selectedPlayer' has selected this object.

name A user-friendly name for this object.

3 Events

3.1 eventGameInit()

An event that is run once as the game is initialized. Not all game may have been properly initialized by this time, so use this only to initialize script state.

3.2 eventStartLevel()

An event that is run once the game has started and all game data has been loaded.

3.3 eventDroidIdle(droid)

A droid should be given new orders.

3.4 eventDroidBuilt(droid[, structure])

An event that is run every time a droid is built. The structure parameter is set if the droid was produced in a factory.

3.5 eventDroidBuilt(structure[, droid])

An event that is run every time a structure is produced. The droid parameter is set if the structure was built by a droid.

3.6 eventAttacked(victim, attacker)

An event that is run when an object belonging to the script's controlling player is attacked. The attacker parameter may be either a structure or a droid.

3.7 eventResearched(research[, structure])

An event that is run whenever a new research is available. The structure parameter is set if the research comes from a research lab.

4 Globals

me The player the script is currently running as.

selectedPlayer The player ontrolled by the client on which the script runs.

gameTime The current game time. Updated before every invokation of a script.

difficulty The currently set campaign difficulty, or the current AI's difficulty setting. It will be one of EASY, MEDIUM, HARD or INSANE.

mapName The name of the current map.

baseType The type of base that the game starts with. It will be one of CAMP_CLEAN, CAMP_BASE or CAMP_WALLS.

alliancesType The type of alliances permitted in this game. It will be one of NO_ALLIANCES, ALLIANCES or ALLIANCES_TEAMS.

powerType The power level set for this game.

maxPlayers The number of active players in this game.

scavengers Whether or not scavengers are activated in this game.

mapWidth Width of map in tiles.

mapHeight Height of map in tiles.

5 Functions

5.1 setTimer(function, milliseconds[, object])

Set a function to run repeated at some given time interval. The function to run is the first parameter, and it must be quoted, otherwise the function will be inlined. The second parameter is the interval, in milliseconds. A third, optional parameter can be a game object to pass to the timer function. If the game object dies, the timer stops running. The minimum number of milliseconds is 100, but such fast timers are strongly discouraged as they may deteriorate the game performance.

```
function conDroids()
{
    ... do stuff ...
}
// call conDroids every 4 seconds
setTimer("conDroids", 4000);
```

5.2 removeTimer(function)

Removes an existing timer. The first parameter is the function timer to remove, and its name must be quoted.

5.3 queue(function[, object])

Queues up a function to run at a later game frame. This is useful to prevent stuttering during the game, which can happen if too much script processing is done at once. The function to run is the first parameter, and it must be quoted, otherwise the function will be inlined. A second, optional parameter can be a game object to pass to the queued function. If the game object dies before the queued call runs, nothing happens.

5.4 bind(function, object[, player])

Bind a function call to an object. The function is called before the object is destroyed. The function to run is the first parameter, and it

must be quoted, otherwise the function will be inlined. The second argument is the object to bind to. A third, optional player parameter may be passed, which may be used for filtering, depending on the object type. *NOTE: This function is under construction and is subject to total change!*

5.5 include(file)

Includes another source code file at this point. This is experimental, and breaks the lint tool, so use with care.

5.6 label(key)

Fetch something denoted by a label. Labels are areas, positions or game objects on the map defined using the map editor and stored together with the map. The only argument is a text label. The function returns an object that has a type variable defining what it is (in case this is unclear). This type will be one of DROID, STRUCTURE, FEATURE, AREA and POSITION. The AREA has defined 'x', 'y', 'x2', and 'y2', while POSITION has only defined 'x' and 'y'.

5.7 enumGroup(group)

Return an array containing all the droid members of a given group.

5.8 newGroup()

Allocate a new group. Returns its numerical ID.

5.9 pursueResearch(lab, research)

Start researching the first available technology on the way to the given technology. First parameter is the structure to research in, which must be a research lab. The second parameter is the technology to pursue, as a text string as defined in "research.txt". The second parameter may also be an array of such strings. The first technology that has not yet been researched in that list will be pursued.

5.10 getResearch(research)

Fetch information about a given technology item, given by a string that matches its definition in "research.txt".

5.11 enumResearch()

Returns an array of all research objects that are currently and immediately available for research.

5.12 componentAvailable(component type, component name)

Checks whether a given component is available to the current player.

5.13 addDroid(player, x, y, name, body, propulsion, reserved, droid type, turrets...)

Create and place a droid at the given x, y position as belonging to the given player, built with the given components. Currently does not support placing droids in multiplayer, doing so will cause a desync.

5.14 buildDroid(factory, name, body, propulsion, reserved, droid type, turrets...)

Start factory production of new droid with the given name, body, propulsion and turrets. The reserved parameter should be passed an empty string for now. The components can be passed as ordinary strings, or as a list of strings. If passed as a list, the first available component in the list will be used. The droid type must be set to match the type of turret passed in.

5.15 enumStruct([player[, structure type[, looking player]]])

Returns an array of structure objects. If no parameters given, it will return all of the structures for the current player. The second parameter is the name of the structure type, as defined in "structures.txt". The third parameter can be used to filter by visibility, the default is not to filter.

5.16 enumFeature(player, name)

Returns an array of all features seen by player of given name, as defined in "features.txt". If player is -1, it will return all features irrespective of visibility to any player. If name is empty, it will return any feature.

5.17 enumDroid([player[, droid type[, looking player]]])

Returns an array of droid objects. If no parameters given, it will return all of the droids for the current player. The second, optional parameter is the name of the droid type, which can currently only be DROID_CONSTRUCT. The third parameter can be used to filter by visibility - the default is not to filter.

5.18 debug(string...)

Output text to the command line.

5.19 pickStructLocation(droid, structure type, x, y)

Pick a location for constructing a certain type of building near some given position. Returns a position object containing "x" and "y" values, if successful.

5.20 structureIdle(structure)

Is given structure idle?

5.21 removeStruct(structure)

Immediately remove the given structure from the map.

5.22 console(strings...)

Print text to the player console.

5.23 groupAddArea(group, x1, y1, x2, y2)

Add any droids inside the given area to the given group.

5.24 groupAddDroid(group, droid)

Add given droid to given group.

5.25 distBetweenTwoPoints(x1, y1, x2, y2)

Return distance between two points.

5.26 groupSize(group)

Return the number of droids currently in the given group.

5.27 droidCanReach(droid, x, y)

Return whether or not the given droid could possibly drive to the given position. Does not take player built blockades into account.

5.28 orderDroidObj(droid, order, object)

Give a droid an order to do something to something.

5.29 orderDroidStatsLoc(droid, order, structure type, x, y)

Give a droid an order to build someting at the given position.

5.30 orderDroidLoc(droid, order, x, y)

Give a droid an order to do something at the given location.

5.31 setMissionTime(time)

5.32 setReinforcementTime(time)

5.33 setStructureLimits(structure type, limit)

5.34 centreView(x, y)

Center the player's camera at the given position.

5.35 playSound(sound[, x, y, z])

Play a sound, optionally at a location.

5.36 gameOverMessage(won)

End game in victory or defeat.

5.37 completeResearch(research[, player])

Finish a research for the given player.

5.38 enableResearch(research[, player])

Enable a research for the given player, allowing it to be researched.

5.39 setPower(power[, player])

Set a player's power directly. (Do not use this in an AI script.)

5.40 enableStructure(structure type)

5.41 addReticuleButton(button type)

5.42 applyLimitSet()

5.43 enableComponent(component, player)

5.44 makeComponentAvailable(component, player)

5.45 allianceExistsBetween(player, player)

5.46 _(string)

Mark string for translation.

5.47 playerPower(player)

Return amount of power held by given player.

5.48 isStructureAvailable(structure type, player)

5.49 isVTOL(droid)

Returns true if given droid is a VTOL (not including transports).

5.50 safeDest(player, x, y)

Returns true if given player is safe from hostile fire at the given location, to the best of that player's map knowledge.

5.51 hackNetOff()

Turn off network transmissions. FIXME - find a better way.

5.52 hackNetOn()

FIXME - find a better way.

6 Gotchas / Bugs

6.1 Object states

Most object states that are changed from the scripts are not in fact changed, but queued up to be synchronized among clients. This means that you cannot check the state later in the same script invokation and expect it to have changed. Instead, if for example you want to mark droids that have been ordered to do something, you can mark them by adding a custom property. Note that this property will not be remembered when it goes out of scope.

6.2 Early research

You cannot set research topics for research labs directly from eventStartLevel. Instead, queue up a function call to set it at some later frame.

6.3 Cyborg construction

Cyborg components are inter-linked, and cannot be passed in as lists as you can with ordinary droids, even though they might look that way.