Starcraft Environment Manual

12th June, 2017

Contents

Chapter 1

Environment

This section will explain how to set up and start a bot with the StarCraft environment using the GOAL programming language.

1.1 Installation

For full installation instructions, see: https://github.com/eishub/StarCraft/wiki/Install-Guide

1.2 Chaoslauncher

In order to make use of all the StarCraft Brood War plugins, you can make use of the Chaoslauncher application. With this application, several plugins can be used like the *BWAPI Injector* which is necessary for using the BWAPI library. It is also recommended to make use of the *APMAlert* plugin, which shows the current actions per minute of all your units together. When the APM of your bot is suddenly very high, your agents might be executing too many actions in a row. It is also recommended to make use of the *W-Mode* plugin. This plugin automatically starts your StarCraft game in windowed mode which is easier for debugging. Yu can also make use of the *ChaosPlugin* to make use of its autoreplay function which automatically saves a replay at the end of each game. You can play these replays by first turning off the *BWAPI Injector*. You can then start StarCraft (in the launcher) and select *Single Player* with gametype *Expansion*. Press the 'Ok' button and then the 'Load Replay' button. If you then open the `Autoreplay` directory in that screen you should be able to see all the replays which are saved by the autoreplay function.

1.3 The Mas2g

The StarCraft environment offers multiple parameters to be set up in the mas2g. Within the mas2g you can specify which map you want to play, specify your own race, set the location of your StarCraft game, turn the development tool on or off, enable the automenu script, and specify which race you want to play against. When any of these parameters are updated, do not forget to close the Chaoslauncher before launching the mas2g, or else your changes will not be applied.

```
use "../../StarCraft Connector.jar" as environment with
    map="(2)Destination.scx",
    own_race="terran",
    StarCraft_location="C:\\StarCraft",
    debug="true",
    auto_menu="Single_Player",
    enemy_race="zerg",
    game_speed=50.
```

1.3.1 Map

It is possible to specify which map the Chaoslauncher will automatically load when starting the game. This can be done by inserting the following line: *map = <filename>*, where *<filename>* is the exact filename of the map (with extension). Please note that the environment only supports maps in the directory: *StarCraft/maps/sscai*. Please note that the first time running the environment on a certain map will take some time (around 2 minutes) to generate the API data of the given map.

1.3.2 Own Race

You may also specify the race of your bot in the mas2g. This will automatically launch the Chaoslauncher with the specified race. You can do this by inserting the following line: *own_race = <RaceName>*, where *<RaceName>* can either be *zerg*, *protoss*, *terran* or *random*. The option *random* will choose one race with a 1/3 chance for each race.

1.3.3 StarCraft Location

It is also possible to specify the location of the StarCraft game. When using the StarCraft game provided by the environment installer, this feature will automatically start the Chaoslauncher when launching the GOAL MAS. When the Chaoslauncher is already running, it will not start again until you close it, but this is fine as long as you use the same init parameters. When the Choaslauncher is automatically started by the environment, an automatic script will be written with all the necessary information to run the GOAL agents (so it is recommended to use this feature). You can use this feature by inserting the line: *StarCraft_location = <FilePath>*, where *<FilePath>* is the absolute path to the StarCraft installation folder.

1.3.4 Debug

The environment also offers a development tool for debugging purposes. With this development tool, you can increase or decrease the game speed, enable cheats and draw unit and map details on the screen. More information about the development tool can be found at 1.4. In order to enable or disable launching the development tool, you can insert the following line: *debug=<Boolean>*.

1.3.5 Auto Menu

The auto menu parameter can be used to automatically go through the menus of the game when starting your agents. This can be used for single player games and multi player games. To use the auto menu function you can insert the following line: *auto_menu=<MenuChoice>*, where *<MenuChoice>* can take the following values:
Single_Player: for a single player game.
Multi_Player: for a multiplayer game.
LAN: for a local multiplayer game.

1.3.6 Enemy Race

The enemy race parameter can be used for specifying which race you want to play against. When an actual enemy race is chosen like: *zerg*, *protoss* or *terran*, the *enemyRace* percept will indicate against which race you are playing. If you do not specify an enemy race, which is equal to the *random* option, the *enemyRace* percept will be *unknown* until the opponent is scouted for the first time. To use the enemy race parameter you can insert the following line: *enemy_race=<RaceName>*, where *<RaceName>* can either be *zerg*, *protoss*, *terran* or *random*. The option *random* will choose one race with a 1/3 chance for each race.

1.3.7 Game Speed

The game speed parameter can be used to set the initial speed of the game when the StarCraft game is launched. StarCraft makes use of a logical frame rate, which means that the game_speed depends on the amount of frames per second (fps) used to update the game. So the higher the fps, the faster the game will go. For using the game_speed parameter you can insert the following line: *game_speed=<FPS>*, where *<FPS>* is a positive integer. If the integer 0 is used, there will be no limit on the amount of FPS used and the game will thus run as fast as it possibly can. **Please note that when integer 0 is used the gameSpeed/1 percept will not give accurate results.** The default (tournament-speed) FPS is 50.

1.3.8 Invulnerable

The invulnerable parameter can be used to make your units invulnerable from the start of the game. This can come in handy for testing purposes when you don't want to fight your opponent. To use the invulnerable function you can insert the following line: *invulnerable=<Boolean>*.

1.3.9 Entity Types

When defining a launch rule it is important that a correct entity type is used. This value has to be the same type of the StarCraft unit without spaces and where the first letter is uncapitalised. So when you for example want to connect an agent to a `terran SCV`, this can be done by using the entity type *terranSCV*. Note that each unit type starts with the race of the unit, followed by the exact name of the unit type.

```
define myAgent as agent {
    use MyAgentInit as init module.
    use MyAgent as main module.
    use MyAgentEvent as event module.
}

launchpolicy {
    when type = terranSCV launch myAgent.
}
```

When using mind control while being protoss, some units from other races can be taken over. These units will also get an entity. An easy way to accomodate all these entities is:

```
while type=* launch ...
```

1.4 The Development Tool

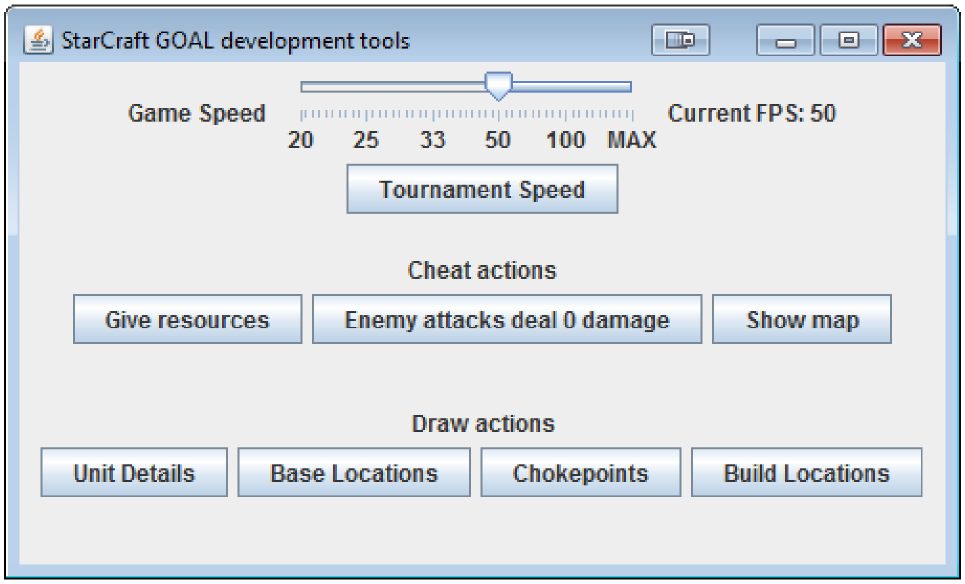

Figure 1.1: Example of the Development Tool

1.4.1 Game Speed

The Game Speed slider can be found at the top of the development tool window. This can be used to quickly change the speed of the game. The initial game speed is set to 50 fps (logical frames). The slowest speed is 20 fps and from there you can set it as fast as you want. Please note that the agent is supposed to play normally at 50 fps which is the default game speed for AI tournaments. When the speed is set to a 100 fps or higher, the agents can react slower than they would on the tournament gamespeed. Setting the game speed on 100 or higher should only be used for quick testing purposes.

1.4.2 Cheat Actions

The development tool offers 3 buttons which instantly enable StarCraft cheats. Note that these cheats should be used for testing purposes only. The first cheat is called: *Give resources* which gives the player 10000 minerals and 10000 gas. The second cheat is called: *Enemy attacks deal 0 damager* which makes the units of the player immune for damage. The last cheat is called: *Show map* which makes the whole map visible for the player. Note that all your agents will then also perceive everything on the map.

1.4.3 Map Drawing

The development tool can also be used to show map or unit details. There are 4 buttons which can be used. First there is the *Unit Details* button which shows the health and *ID* of every unit. There is also the *Base Locations* button which shows all the starting locations of the map and also all the base locations on the map where players could be expanding to. There is also the *Chokepoints* button which shows all the chokepoints (which are the narrow points where not many units can go through at the same time) on the map. Finally there is the *Build Locations* button which shows all the non-obstructed and explored building locations of the map which the worker units perceive with the *constructionSite* percept.

Chapter 2

Percepts

This section will list all the percepts that are usable in the StarCraft environment. The percepts vary per unit, for example: an attacking unit will not perceive the amount of resources available to the player as he does not need them. For the implementation of these percepts in your GOAL code, please refer to the GOAL programming guide.

2.1 Percepts for All Units and Buildings

2.1.1 Available Resources

Resources percept

Description	The amount of minerals, gas and supply available to the player (i.e. shared by all units). NOTE: supply is multiplied by 2 throughout this interface, so 10 supply in game corresponds with 20 supply in this environment.
Type	Send on change
Syntax	`resources(<M>, <G>, <CS>, <TS>)`
Example	`resources(350, 100, 25, 41)`

Parameters

`<M>`	The current amount of minerals available to the player.
Type	`Positive Integer`
Range	[0–∞]
`<G>`	The current amount of gas available to the player.
Type	`Positive Integer`
Range	[0–∞]
`<CS>`	The supply of the player which is currently in use.
Type	`Positive Integer`
Range	[0–400]
`<TS>`	The total amount of supply the player can currently use. Note that `<TS>` is always greater or equal to `<CS>`
Type	`Positive Integer`
Range	[0–400]

2.1.2 Unit Information

Self percept

Description	The (unique) *ID* and type of the unit. Also gives information about the maximum health, shield and energy of the unit.
Type	Send once
Syntax	`self(<ID>, <UnitType>, <MaxHealth>, <MaxShield>, <MaxEnergy>)`
Example	`self(21, Terran SCV, 60, 0, 0)`

Parameters

`<ID>`	The (unique) *ID* of the unit.
Type	`Positive Integer`
Range	[0–∞]
`<UnitType>`	The type of the unit. The type of a unit consists of a string with the race of the unit and the name of the unit parted by a space. See Section 6 for the list of all the unit types.
Type	`String`
`<MaxHealth>`	The maximum amount of health of the unit.
Type	`Positive Integer`
Range	[0–2500]
`<MaxShield>`	The maximum amount of shield of the unit.
Type	`Positive Integer`
Range	[0–2500]
`<MaxEnergy>`	The maximum amount of energy of the unit.
Type	`Positive Integer`
Range	[0–2500]

Defensive Matrix percept

Description	Information about how much health the defensive matrix has left on the unit. Note: this only applies to specific Terran units.
Type	Send on change
Syntax	`defensiveMatrix(<health>)`
Example	`defensiveMatrix(200)`

Parameters

`<health>`	The amount of health left of the defensive matrix.
Type	`Positive Integer`
Range	[0–250]

Status percept

Description	The current amount of health, shield and energy of the unit. The `status` percept also shows the conditions of the unit and its current position.
Type	Send on change
Syntax	`status(<Health>, <Shield>, <Energy>, <Cond>, <X>, <Y>)`
Example	`status(250, 0, 0, [moving, carrying], 24, 36)`

Parameters

`<Health>`	The current amount of health of the unit.
Type	`Positive Integer`
Range	[0–`<MaxHealth>`] where `<MaxHealth>` is the maximum health of the given unit.
`<Shield>`	The current amount of shields of the unit.
Type	`Positive Integer`
Range	[0–`<MaxShield>`] where `<MaxShield>` is the maximum shield of the given unit.
`<Energy>`	The current amount of energy of the unit.
Type	`Positive Integer`
Range	[0–`<MaxEnergy>`] where `<MaxEnergy>` is the maximum energy of the given unit.
`<Cond>`	The current condition of the unit. Each unit can have multiple or no conditions depending on the unit and situation. See Section 2.4 for the list of all the conditions.
Type	`List of Strings`
`<X>`	The x-coordinate of the unit in the map.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit in the map.
Type	`Positive Integer`
Range	[0–∞]

New Unit percept

Description	Indicates when a new unit is under construction.
Type	Send on change
Syntax	`newUnit(<ID>,<X>,<Y>)`
Example	`newUnit(44, 22, 37)`

Parameters		
	<ID>	The (unique) *ID* of the unit.
	Type	`Positive Integer`
	Range	[0–∞]
	<X>	The x-coordinate of the unit in the map.
	Type	`Positive Integer`
	Range	[0–∞]
	<Y>	The y-coordinate of the unit in the map.
	Type	`Positive Integer`
	Range	[0–∞]

2.1.3 Player Percepts

Enemy Race percept

Description	The race of your opponent.
Type	Send once
Syntax	`enemyRace(<Race>)`
Example	`enemyRace(protoss)`
Example	When playing against a random race.

```
if bel(enemyRace(unknown), enemy(Type,_,_,_,_),
sub_string(Type, 0,1,_,Race)) then {
  if bel(Race = "Z") then delete(enemyRace(unknown))
  + insert(enemyRace(zerg)).
  if bel(Race = "P") then delete(enemyRace(unknown))
  + insert(enemyRace(protoss)).
  if bel(Race = "T") then delete(enemyRace(unknown))
  + insert(enemyRace(terran)).
}
```

Parameters		
	<Race>	The enemy race which can take the value: protoss, terran, zerg or unknown when the enemy race is random.
	Type	`String`

Game Speed percept

Description	The current game speed in frames per seconds. For more information see Section 1.3.7.
Type	Send on change
Syntax	`gameSpeed(<FPS>)`
Example	`gameSpeed(50)`

Parameters

`<FPS>`	The current amount of frames per second the game is updated by.
Type	`Positive Integer`
Range	[0–∞]

Frame percept

Description	The current game frame, sent per 50 frames (which is also the interval at which construction sites are updated). For more information see Section 1.3.7.
Type	Send on change
Syntax	`frame(<number>)`
Example	`frame(150)`

Parameters

`<number>`	The game frame in increments of 50.
Type	`Positive Integer`
Range	[0–∞]

Winner percept

Description	At the end of the game all units will receive this depending whether they have won or not.
Type	Send always
Syntax	`winner(<hasWon>)`
Example	`winner(true)`

Parameters

`<hasWon>`	Boolean which indicates whether you have won or not.
Type	`Boolean`

Nuke percept

Description	Indicates that a nuclear strike will land on the given position.
Type	Send on change
Syntax	`nuke(<X>,<Y>)`
Example	`nuke(22, 37)`

Parameters

`<X>`	The x-coordinate of the nuclear strike.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the nuclear strike.
Type	`Positive Integer`
Range	[0–∞]

2.1.4 Map Percepts

Map percept

Description	The width and the height of the map (no. of squares).
Type	Send once
Syntax	`map(<Width>,<Height>)`
Example	`map(96, 128)`
Parameters	

`<Width>`	The width of the map.
Type	`Positive Integer`
Range	[0–∞]
`<Height>`	The height of the map.
Type	`Positive Integer`
Range	[0–∞]

Base percept

Description	All the base locations of the map. These are possible construction sites for bases.
Type	Send once
Syntax	`base(<X>,<Y>,<IsStart>,<ResourceGroupID>)`
Example	`base(28, 32, true, 8)`
Parameters	

`<X>`	The x-coordinate of the base location.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the base location.
Type	`Positive Integer`
Range	[0–∞]
`<IsStart>`	Indicates whether the location is a starting location or not.
Type	`Boolean (true or false)`
`<ResourceGroupID>`	The Resource Group that is closest to this base location. The vespene geyser and all mineral fields will share this Resource group.
Type	`Positive Integer`
Range	[0–∞]

Chokepoint/3 percept

Description	All the chokepoints on the map. These are the narrow points on the map where only a limited amount of units can go through at the same time.
Type	Send once
Syntax	`chokepoint(<X>,<Y>,<W>)`
Example	`chokepoint(12, 15, 50)`

Parameters

`<X>`	The x-coordinate of the chokepoint.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the chokepoint.
Type	`Positive Integer`
Range	[0–∞]
`<W>`	The size of the chokepoint (in pixels).
Type	`Positive Integer`
Range	[0–∞]

Chokepoint/4 percept

Description	All the chokepoints on the map. These are the narrow points on the map where only a limited amount of units can go through at the same time.
Type	Send once
Syntax	`chokepoint(<X1>,<Y1>,<X2>,<Y2>)`
Example	`chokepoint(12, 15, 14, 17)`

Parameters

	`<X1>`	The x-coordinate of the first side of the chokepoint.
	Type	`Positive Integer`
	Range	[0–∞]
	`<Y1>`	The y-coordinate of the first side of the chokepoint.
	Type	`Positive Integer`
	Range	[0–∞]
	`<X2>`	The x-coordinate of the second side of the chokepoint.
	Type	`Positive Integer`
	Range	[0–∞]
	`<Y2>`	The y-coordinate of the second side of the chokepoint.
	Type	`Positive Integer`
	Range	[0–∞]

Chokepoint/6 percept

Description	All the chokepoints on the map. These are the narrow points on the map where only a limited amount of units can go through at the same time. This percept also sends the regions that this chokepoint connects.
Type	Send once
Syntax	`chokepoint(<X1>,<Y1>,<X2>,<Y2>,<RegionID1>,<RegionID2>)`
Example	`chokepoint(12, 15, 14, 17, 1, 2)`

Parameters

`<X1>`	The x-coordinate of the first side of the choke-point.
Type	`Positive Integer`
Range	[0–∞]
`<Y1>`	The y-coordinate of the first side of the choke-point.
Type	`Positive Integer`
Range	[0–∞]
`<X2>`	The x-coordinate of the second side of the choke-point.
Type	`Positive Integer`
Range	[0–∞]
`<Y2>`	The y-coordinate of the second side of the choke-point.
Type	`Positive Integer`
Range	[0–∞]
`<RegionID1>`	The ID of the first region.
Type	`Positive Integer`
Range	[0–∞]
`<RegionID2>`	The ID of the second region.
Type	`Positive Integer`
Range	[0–∞]

Region/5 percept

Description	All the regions on the map. Regions are connected by choke-points and can be on high or low ground.
Type	Send once
Syntax	`region(<Id>,<CenterX>,<CenterY>,<Height>,<ConnectedRegionsList>)`
Example	`region(12, 15, 14, 17, [1,2])`

Parameters

`<Id>`	The ID of the region.
Type	`Positive Integer`
Range	[0–∞]
`<CenterX>`	The x-coordinate of the center of the region.
Type	`Positive Integer`
Range	[0–∞]
`<CenterY>`	The y-coordinate of the center of the region.
Type	`Positive Integer`
Range	[0–∞]
`<Height>`	The height of the region.
Type	`Positive Integer`
Range	[0–∞]
`<ConnectedRegionsList>`	A list of regions that are connected to this region.
Type	`List`

Region/2 percept

Description	Gives information about which region on the map a unit is in.
Type	Send on change
Syntax	`region(<UnitID>,<RegionID>)`
Example	`region(12, 15)`

Parameters

`<UnitID>`	The Id of a unit in a region. Sent for every visible unit.
Type	`Positive Integer`
Range	[0–∞]
`<RegionID>`	The ID of the region.
Type	`Positive Integer`
Range	[0–∞]

2.1.5 Unit Percepts

Friendly percept

Description	Information about all (alive) units of the player.
Type	Send always
Syntax	`friendly(<Type>,<ID>,<Condition>)`
Example	`friendly(Protoss Gateway, 26, [beingConstructed])`

Parameters

`<Type>`	The type of the unit. The type of a unit consists of a string with the race of the unit and the name of the unit parted by a space. See Section 6 for the list of all the unit types.
Type	`String`
`<ID>`	The (unique) *ID* of the unit.
Type	`Positive Integer`
Range	[0–∞]
`<Cond>`	The current condition of the unit. Each unit can have multiple or no conditions depending on the unit and situation. See Section 2.4 for the list of all actual conditions.
Type	`List of Strings`

Enemy percept

Description	Information about all (alive) enemy units that are currently visible to the player.
Type	Send always
Syntax	`enemy(<Type>,<ID>,<Health>,<Shield>,<Condition>,<X>,<Y>)`
Example	`enemy(Zerg Overlord, 12, 200, 0, [flying], 120, 96)`
Note	`Enemy units that are cloaked are also percepted, but cannot be attacked.`
Parameters	

`<Type>`	The type of the unit. The type of a unit consists of a string with the race of the unit and the name of the unit parted by a space.
Type	`String`
`<ID>`	The (unique) *ID* of the unit.
Type	`Positive Integer`
Range	[0–∞]
`<Health>`	The current amount of health of the unit.
Type	`Positive Integer`
Range	[1–`<maxHealth>`] where `<maxHealth>` is the maximum health of the given unit.
`<Shield>`	The current amount of shields of the unit.
Type	`Positive Integer`
Range	[0–`<maxShield>`] where `<maxShield>` is the maximum shield of the given unit.
`<Cond>`	The current condition of the unit. Each unit can have multiple or no conditions depending on the unit and situation. Not all the conditions are available for the enemy percept. See Section 2.4 for the list of all actual conditions.
Type	`List of Strings`
`<X>`	The x-coordinate of the unit.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit.
Type	`Positive Integer`
Range	[0–∞]

Attacking percept

Description	Shows the enemy units which are attacking and which units they have targeted.
Type	Send always
Syntax	`attacking(<ID>,<TargetID>)`
Example	`attacking(123, 177)`

Parameters

`<ID>`	The (unique) *ID* of the enemy unit which is attacking.
Type	`Positive Integer`
Range	[0–∞]
`<TargetID>`	The (unique) ID of the targeted unit which is being attacked.
Type	`Positive Integer`
Range	[0–∞]

2.2 Building Percepts

2.2.1 Research and Upgrade Percepts

Researching percept

Description	Indicates which *tech* is currently being researched. The research is done when this percept is no longer seen. See Section 4 for the list of tech types.
Type	Send always
Syntax	`researching(<TechType>)`
Example	`researching(Stim Packs)`

Parameters

`<TechType>`	The *tech* which is currently researched.
Type	`String`

Upgrading percept

Description	Indicates which *upgrade* is currently being performed. The upgrade is done when this percept is no longer seen. See Section 5 for the list of all actual tech types.
Type	Send always
Syntax	`upgrading(<UpgradeType>)`
Example	`upgrading(Ocular Implants)`

Parameters

`<UpgradeType>`	The *upgrade* which is currently upgraded.
Type	`String`

2.2.2 Production Buildings

Queue Size percept

Description	Shows how many units are in queue of the production building. Hatchery: Shows the amount of available larva units.
Type	Send on change
Syntax	`queueSize(<Size>)`
Example	`queueSize(2)`
Parameters	

`<Size>`	The size of the current queue.
Type	`Positive Integer`
Range	[0–5]

Rally Point percept

Description	The position of the building's rallypoint in map coordinates.
Type	Send on change
Syntax	`rallyPoint(<X>,<Y>)`
Example	`rallyPoint(76, 45)`
Parameters	

`<X>`	The x-coordinate of the rallypoint.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the rallypoint.
Type	`Positive Integer`
Range	[0–∞]

Rally Unit percept

Description	Shows on which unit the building's rallypoint is set.
Type	Send on change
Syntax	`rallyUnit(<UnitID>)`
Example	`rallyUnit(145)`
Parameters	

`<UnitID>`	The (unique) *ID* the rallypoint points to.
Type	`Positive Integer`
Range	[0–∞]

2.2.3 Loadable Buildings

Space Provided percept

Description	Shows how many units are currently loaded in the building and how the maximun amount of units that can be loaded in the building.
Type	Send on change
Syntax	`spaceProvided(<CSize>, <MSize>)`
Example	`spaceProvided(2, 4)`

Parameters

`<CSize>`	The amount of currently loaded units.
Type	`Positive Integer`
Range	[0–∞]
`<MSize>`	The maximum amount of units that can be loaded.
Type	`Positive Integer`
Range	[0–∞]

Unit Loaded percept

Description	Shows which unit is loaded inside the given loadable unit.
Type	Send always
Syntax	`unitLoaded(<ID>, <Type>)`
Example	`unitLoaded(154, Terran Marine)`

Parameters

`<ID>`	The (unique) *ID* of the loaded unit.
Type	`Positive Integer`
Range	[0–∞]
`<Type>`	The type of the loaded unit.
Type	`String`

2.3 Worker percepts

2.3.1 Worker Management

Worker Activity Percept

Description	Shows the current activity of the worker.
Type	Send on change
Syntax	`workerActivity(<Activity>)`
Example	`workerActivity(gatheringGas)`

Parameters

`<Activity>`	The current activity of the worker unit. Can take values: gatheringGas, gatheringMinerals, constructing or idling.
Type	`String`

2.3.2 Builder Percepts

Vespene Geyser percept

Description	Information about a visible (possibly empty) vespene geyser on the map.
Type	Send always
Syntax	`vespeneGeyser(<ID>,<Resources>,<ResourceGroup>,<X>,<Y>)`
Example	`vespeneGeyser(57, 5000, 6, 22, 32)`

Parameters

`<ID>`	The (unique) *ID* of the vespene geyser.
Type	`Positive Integer`
Range	[0–∞]
`<Resources>`	The amount of resources left in the vespene geyser.
Type	`Positive Integer`
Range	[0–5000]
`<ResourceGroup>`	The resource group of the vespene geyser.
Type	`Positive Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the vespene geyser.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the vespene geyser.
Type	`Positive Integer`
Range	[0–∞]

Mineral Field percept

Description	Information about a visible (non-empty) mineralfield on the map.
Type	Send always
Syntax	`mineralField(<ID>,<Resources>,<ResourceGroup>,<X>,<Y>)`
Example	`mineralField(57, 5000, 6, 22, 32)`
Parameters	

`<ID>`	The (unique) *ID* of the mineralfield.
Type	`Positive Integer`
Range	[0–∞]
`<Resources>`	The amount of resources left in the mineralfield.
Type	`Positive Integer`
Range	[1–5000]
`<ResourceGroup>`	The resource group of the mineralfield.
Type	`Positive Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the mineralfield.
Type	`Positive Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the mineralfield.
Type	`Positive Integer`
Range	[0–∞]

Construction Site percept

Description	Shows all explored and non-obstructed construction sites where at least a Protoss Nexus, Terran Command Center or Zerg Hatchery can be placed. Updated every 50 game frames.
Type	Send always
Syntax	(If Protoss) `constructionSite(<X>,<Y>,<InPylonRange>)` (If Zerg) `constructionSite(<X>,<Y>,<OnCreep>)` (If Terran) `constructionSite(<X>,<Y>)`

Example	`constructionSite(66, 98, false)`
	`constructionSite(66, 98)`

Parameters		
	`<X>`	The x-coordinate of the construction site.
	Type	`Positive Integer`
	Range	[0–∞]
	`<Y>`	The y-coordinate of the construction site.
	Type	`Positive Integer`
	Range	[0–∞]
	`<InPylonRange>`	Indicates whether the construction site is in range of a pylon (this is only for Protoss)
	Type	`Boolean`
	Range	[True-False]
	`<OnCreep>`	Indicates whether the construction site is on creep (this is only for Zerg)
	Type	`Boolean`
	Range	[True-False]

2.4 Conditions

2.4.1 Worker Units

`carrying`	Indicates when the worker unit is carrying minerals or vespene gas.
`constructing`	Shows that the worker unit is busy constructing a building.

2.4.2 Buildings

`beingConstructed`	Indicates when a building is being constructed.
`lifted`	Indicates when the building is lifted.
`<addonName>`	Indicates when an addon of the building is present, gives the exact addonname.

2.4.3 All

`idle`	Indicates when the unit is idle (not doing anything).
`cloaked`	Indicates when a unit is cloaked.
`moving`	Shows that a unit is moving.
`following`	Shows that a unit is following an other unit.
`loaded`	Indicates when a unit is loaded.
`defenseMatrixed`	Indicates when a unit has a defense matrix on it.
`flying`	Shows that a unit is flying.
`patrolling`	Shows that a unit is patrolling between 2 positions.
`underAttack`	Indicates when the unit is under attack.
`attacking`	Indicates when a unit is attacking an other unit.
`coolingDown`	Indicates when a unit cannot attack due to cooldown.
`detected`	Indicates when an enemy cloaked/burrowed unit has been detected (and thus can be attacked).

2.4.4 Zerg

`burrowed`	Indicates when a zerg unit is burrowed.
`ensnared`	Shows that the unit is ensnared by a Queen unit.
`parasited`	Shows that the unit is parasited by a Queen unit.
`plagued`	Indicates that the unit is plagued by a Defiler unit.
`darkSwarmed`	Indicates that the unit is under a Dark Swarm from a Defiler unit.
`acidSpored`	Indicates when a unit is under a Acid Spores from a Defiler unit.
`morphing`	Indicates that a unit is morphing.

2.4.5 Terran

`stimmed`	Indicates when a firebat or marine is stimmed.
`sieged`	Indicates when a siegetank is in siegemode.
`blinded`	Shows when a unit is blinded by a medic.
`lockDowned`	Indicates when a unit is under lockdown by a Ghost unit.
`irradiated`	Shows when a unit is irradiated by a Science Vessel.
`nukeReady`	Indicates when a unit is ready to use Nuclear Missile.
`beingHealed`	Indicates when a unit is healed or repaired by another unit.
`repairing`	Shows that a unit is repairing.
`hasMines`	Indicates whether a vulture has mines.

2.4.6 Protoss

`underStorm`	Shows when a unit is under a storm from a High Templar unit.
`stasised`	Indicates when a unit is stuck in stasis.
`maelstrommed`	Indicates when a unit is maelstrommed by a Dark Archon.
`disruptionWebbed`	Shows when a unit is in a disruption web from a Corsair.
`hasScarabs`	Indicates if a reaver has scarabs.

Chapter 3

Actions

This section will list all the actions that are usable in the Starcraft environment.

3.1 Attack action

Description	Attack the given enemy unit.
Syntax	`attack(<TargetID>)`
Parameters	`<TargetID>`: The *ID* of the target that will be attacked.
Pre	The targeted unit is attack capable.
Post	The targeted unit is being attacked by your unit.
Note	Medics that use this action on a friendly unit will heal them. Medics cannot attack enemies.

3.2 Move action

Description	Move to the chosen location.
Syntax	`move(<X>,<Y>)`
Parameters	`<X>`: The x-coordinate of the chosen location `<Y>`: The y-coordinate of the chosen location
Pre	The unit is capable of moving to the chosen location.
Post	The unit moves to the chosen location (ignoring any other unit it might pass by).

3.3 Attack move action

Description	Go to the given location and attack everything you encounter.
Syntax	`attack(<X>,<Y>)`
Parameters	`<X>`: The x-coordinate of the chosen location `<Y>`: The y-coordinate of the chosen location
Pre	The unit is capable of moving to the chosen location.
Post	The unit moves to the chosen locations and attacks any attack capable enemy unit it encounters.

3.4 Upgrade action

Description	Starts working on the chosen upgrade.
Syntax	`upgrade(<UpgradeName>)`
Parameters	`<UpgradeName>`: The name of the upgrade you want to upgrade.
Pre	The unit is capable of upgrading and has sufficient resources to do so.
Post	The unit starts upgrading the chosen upgrade.

3.5 Build action

Desription	Build a building on the given location.
Syntax	`build(<Type>,<X>,<Y>)`
Parameters	`<Type>`: The Type of the building that has to be built. `<X>`: The x-coordinate of the chosen build location `<Y>`: The y-coordinate of the chosen build location
Pre	The unit is capable of constructing the chosen building and the chosen location is not obstructed.
Post	The unit starts constructing the chosen building at the chosen location.

3.6 Gather action

Description	Gather the chosen resource (minerals or vespene gas).
Syntax	`gather(<ID>)`
Parameters	`<ID>`: The *ID* of the chosen resource.
Pre	The unit is capable of performing the gather action and a valid resource unit is selected.
Post	The unit starts gathering the chosen resource.

3.7 Train action

Description	Train the chosen unit.
Syntax	`train(<Type>)`
Parameters	`<Type>`: The type of unit to train.
Pre	The production facility is capable of producing the chosen unit and has sufficient resources to do so.
Post	The production facility starts producing the chosen unit.

3.8 Stop action

Description	Stop performing the current action.
Syntax	`stop`
Pre	The unit is performing some kind of action.
Post	The unit stops performing the action.

3.9 Ability action

Description	Use an (researched) ability.
Syntax	`ability(<Type>)`
Parameters	`<Type>`: The type of technology to use.
Pre	The chosen tech type is researched and the unit is capable of performing the chosen tech type.
Post	The unit performs the chosen tech ability.
Note	In game behaviour that can be toggled on and off such as Burrow/Cloak/Siege is also executed by using this action.

3.10 Ability on target action

Description	Use an (researched) ability on a target.
Syntax	`ability(<Type>, <Target>)`
Parameters	`<Type>`: The type of technology to use.
	`<Target>`: The target to use the technology on.
Pre	The chosen tech type is researched, the unit is capable of performing the chosen tech type and the chosen target is attack capable.
Post	The unit performs the chosen tech ability on the chosen target.

3.11 Ability on location action

Description	use an (researched) ability on a location.
Syntax	`ability(<Type>, <X>, <Y>)`
Parameters	`<Type>`: The type of technology to use.
	`<X>`: The x-coordinate of the chosen location
	`<Y>`: The y-coordinate of the chosen location.
Pre	The chosen tech type is researched, the unit is capable of performing the chosen tech type and the chosen location is valid to perform an action on.
Post	The unit performs the chosen tech ability on the chosen location.

3.12 Research action

Description	Research a chosen tech type.
Syntax	`research(<Type>)`
Parameters	`<Type>`: The type of tech to research.
Pre	The building is capable of researching the chosen tech type and has sufficient resources to do so.
Post	The building starts researching the chosen tech type.

3.13 Set rally point action

Description	Set the rally point on a specific location. When the rally point is set, produced units of this production facility will automatically move to this location.
Syntax	`setRallyPoint(<X>, <Y>)`
Parameters	`<X>`: The x-coordinate of the chosen rally location `<Y>`: The y-coordinate of the chosen rally location.
Pre	The building is capable of setting up a rally point and the chosen location is a valid location where units can move to.
Post	The building sets the rally point on the chosen location.

3.14 Set rally point to unit action

Description	Set the rally point on a unit. When the rally point is set, produced units of this production facility will automatically move to this unit.
Syntax	`setRallyPoint(<Unit>)`
Parameters	`<Unit>`: The unit to set the rally point on.
Pre	The building is capable of setting up a rally point and the chosen unit is on a valid location where units can move to.
Post	The building sets the rally point on the chosen unit.

3.15 Lift action

Description	Lift into the air.
Syntax	`lift`
Pre	The building is capable of flying and is not busy performing any other action.
Post	The building starts flying.
Note	Only for Terran buildings.

3.16 Land action

Desription	Land on the given location.
Syntax	`land(<X>, <Y>)`
Parameters	`<X>`: The x-coordinate of the chosen land location `<Y>`: The y-coordinate of the chosen land location.
Pre	The unit is currently flying and is capable of landing on the chosen location.
Post	The unit lands on the chosen location.
Note	The location has to be visible.

3.17 Build addon action

Desription	Build the chosen addon.
Syntax	`buildAddon(<Name>)`
Parameters	`<Name>`: The name of the chosen addon.
Pre	The building is capable of building the addon and does not already have the addon.
Post	The building starts constructing the addon.
Note	Only for Terran buildings.

3.18 Load action

Desription	Load a unit.
Syntax	`load(<ID>)`
Parameters	`<ID>`: The *ID* of the unit to load into this (loadable) unit.
Pre	The unit is capable of loading other units inside it and still has enough space prodivded for the targeted unit.
Post	The targeted unit starts walking to the loadable unit and loads into it.

3.19 Unload action

Desription	Unload an unit.
Syntax	`unload(<ID>)`
Parameters	`<ID>`: The *ID* of the unit to unload from this (loadable) unit.
Pre	The unit is capable of loading other units inside it.
Post	The targeted unit is unloaded and stands next to the (loadable) unit.

3.20 Unload all action

Desription	Unload all units.
Syntax	`unloadAll`
Pre	The unit is capable of loading other units inside it.
Post	All units are unloaded and stand next to the (loadable) unit.

3.21 Cancel action

Desription	Cancel the construction of the unit.
Syntax	`cancel(<Id>)`
Parameters	`<ID>`: The *ID* of the building/unit to cancel upgrading/morphing etc. Passing no argument will cancel the unit itself.
Pre	The unit is morphing, beingConstructed, researching or upgrading.
Post	The morphing, construction, upgrading or researching is cancelled.

3.22 Patrol action

Desription	Patrol an unit between the place it's standing and the given location.
Syntax	`patrol(<X>, <Y>)`
Parameters	`<X>`: The x-coordinate of the chosen location `<Y>`: The y-coordinate of the chosen location
Pre	The unit is capable of moving to the chosen location.
Post	The unit patrols between the chosen location (ignoring any other unit it might pass by) and the location it was originally standing.
Note	Medics that are patrolling will automatically heal allies in range.

3.23 Morph action (Zerg only)

Desription	Morphs a zerg unit into the another unit.
Syntax	`morph(<Type>)`
Parameters	`<Type>`: The type of unit to morph.
Pre	The unit is capable of morphing into another unit and does have to resources to do so.
Post	The agent terminates and a new agent is created for the new unit.

3.24 Follow action

Description	Follows a unit.
Syntax	`follow(<Id>)`
Parameters	`<Id>`: The Id of the unit to follow.
Pre	The unit is capable of moving.
Post	The unit follows the selected unit.

3.25 Repair action (Terran only)

Description	Repairs a unit with less than the max amount of health. Also works for unfinished buildings.
Syntax	`repair(<Id>)`
Parameters	`<Id>`: The Id of the unit to repair.
Pre	The unit is a SCV, has the resources to repair and can reach the unit to repair.
Post	The SCV starts repairing the selected unit.

3.26 Forfeit action

Description	Forfeits the game.
Syntax	`forfeit()`
Pre	The game is in progress.
Post	The game ends with a loss.

Chapter 4

Tech Types

All the tech types that can be researched for each race.

4.1 Terran Units

4.1.1 Battle Cruisers

```
Yamato Gun
```

4.1.2 Command Centers

```
Scanner Sweep
```

4.1.3 Ghosts

```
Lockdown
Personel Cloaking
Nuclear Strike
```

4.1.4 Marines and Firebats

```
Stim Packs
```

4.1.5 Medics

```
Healing
Restoration
Optical Flare
```

4.1.6 Science Vessels

`Defensive Matrix`
`EMP Shockwave`
`Irradiate`

4.1.7 Siege Tanks

`Tank Siege Mode`

4.1.8 Vultures

`Spider Mines`

4.1.9 Wraith

`Cloaking Field`

4.2 Protoss Units

4.2.1 Arbiters

`Cloaking Field`
`Recall`
`Stasis Field`

4.2.2 Corsairs

`Disruption Web`

4.2.3 Dark Archons

`Feedback`
`Maelstrom`
`Mind Control`

4.2.4 Dark Templars

`Dark Archon Meld`

4.2.5 High Templars

`Archon Warp`
`Psionic Storm`
`Hallucination`

4.3 Zerg Units

4.3.1 Generic

`Burrowing`

4.3.2 Defilers

`Dark Swarm`
`Plague`
`Consume`

4.3.3 Hydralisks

`Lurker Aspect`

4.3.4 Lurkers

`Burrowing` (Can be used without having it researched)

4.3.5 Queens

`Infestation`
`Parasite`
`Ensnare`
`Spawn Broodlings`

Chapter 5

Upgrade Types

All the upgrade types that can be used for each race.

5.1 Terran Units

5.1.1 Academy

```
U-238 Shells
Caduceus Reactor
```

5.1.2 Armory

```
Terran Vehicle Weapons
Terran Vehicle Plating
Terran Ship Weapons
Terran Ship Plating
```

5.1.3 Covert Ops

```
Ocular Implants
Moebius Reactor
```

5.1.4 Engineering Bay

```
Terran Infantry Weapons
Terran Infantry Armor
```

5.1.5 Machine Shop

`Ion Thrusters`
`Charon Boosters`

5.1.6 Physics Lab

`Colossus Reactor`

5.1.7 Science Facility

`Titan Reactor`

5.1.8 Control Tower

`Apollo Reactor`

5.2 Protoss Units

5.2.1 Arbiter Tribunal

`Khaydarin Core`

5.2.2 Citadel of Adun

`Protoss Plasma Shields`
`Leg Enhancements`

5.2.3 Cybernetics Core

`Singularity Charge`
`Protoss Air Weapons`
`Protoss Air Armor`

5.2.4 Fleet Beacon

`Apial Sensors`
`Gravitic Thrusters`
`Argus Jewel`
`Carrier Capacity`

5.2.5 Forge

```
Protoss Plasma Shields
Protoss Ground Armor
Protoss Ground Weapons
```

5.2.6 Observatory

```
Gravitic Boosters
Sensor Array
```

5.2.7 Robotics Support Bay

```
Reaver Capacity
Scarab Damage
Gravitic Drive
```

5.2.8 Templar Archives

```
Argus Talisman
Khaydarin Amulet
```

5.3 Zerg Units

5.3.1 Defiler Mound

```
Metasynaptic Node
```

5.3.2 Evolution Chamber

```
Zerg Melee Attacks
Zerg Missile Attacks
Zerg Carapace
```

5.3.3 Hydralisk Den

```
Muscular Augments
Grooved Spines
```

5.3.4 Lair and Hive

`Ventral Sacs`
`Antennae`
`Pneumatized Carapace`

5.3.5 Queen's Nest

`Gamete Meiosis`

5.3.6 Spawning Pool

`Metabolic Boost`
`Adrenal Glands`

5.3.7 (Greater) Spire

These are the upgrade type(s) the (Greater) Spire offers.
`Zerg Flyer Carapace`
`Zerg Flyer Attacks`

5.3.8 Ultralisk Cavern

These are the upgrade type(s) the Ultralisk Cavern offers.
`Chitinous Plating`
`Anabolic Synthesis`

Chapter 6

Unit Types

StarCraft's unit types.

6.1 Terran Units

6.1.1 Terran Ground Units

```
Terran Firebat
Terran Ghost
Terran Goliath
Terran Marine
Terran Medic
Terran SCV
Terran Siege Tank
Terran Vulture
Terran Vulture Spider Mine
```

6.1.2 Terran Air Units

```
Terran Battlecruiser
Terran Dropship
Terran Science Vessel
Terran Valkyrie
Terran Wraith
```

6.1.3 Terran Building Units

`Terran Academy`
`Terran Armory`
`Terran Barracks`
`Terran Bunker`
`Terran Command Center`
`Terran Engineering Bay`
`Terran Factory`
`Terran Missle Turret`
`Terran Refinery`
`Terran Science Facility`
`Terran Starport`
`Terran Supply Depot`

6.1.4 Terran Addons

`Terran Comsat Station`
`Terran Control Tower`
`Terran Covert Ops`
`Terran Machine Shop`
`Terran Nuclear Silo`
`Terran Physics Lab`

6.2 Protoss Units

6.2.1 Protoss Ground Units

`Protoss Archon`
`Protoss Dark Archon`
`Protoss Dark Templar`
`Protoss Dragoon`
`Protoss High Templar`
`Protoss Probe`
`Protoss Reaver`
`Protoss Scarab`
`Protoss Zealot`

6.2.2 Protoss Air Units

Protoss Arbiter
Protoss Carrier
Protoss Corsair
Protoss Interceptor
Protoss Observer
Protoss Scout
Protoss Shuttle

6.2.3 Protoss Building Units

Protoss Arbiter Tribunal
Protoss Assimilator
Protoss Citadel of Adun
Protoss Cybernetics Core
Protoss Fleet Beacon
Protoss Forge
Protoss Gateway
Protoss Nexus
Protoss Observatory
Protoss Photon Cannon
Protoss Pylon
Protoss Robotics Facility
Protoss Robotics Support Bay
Protoss Shield Battery
Protoss Stargate
Protoss Templar Archives

6.3 Zerg Units

6.3.1 Zerg Ground Units

Zerg Broodling
Zerg Defiler
Zerg Drone
Zerg Egg
Zerg Hydralisk
Zerg Infested Terran
Zerg Larva

`Zerg Lurker`
`Zerg Lurker Egg`
`Zerg Ultralisk`
`Zerg Zergling`

6.3.2 Zerg Air Units

`Zerg Cocoon`
`Zerg Devourer`
`Zerg Guardian`
`Zerg Mutalisk`
`Zerg Overlord`
`Zerg Queen`
`Zerg Scourge`

6.3.3 Zerg Building Units

`Zerg Creep Colony`
`Zerg Defiler Mound`
`Zerg Evolution Chamber`
`Zerg Extractor`
`Zerg Greater Spire`
`Zerg Hatchery`
`Zerg Hive`
`Zerg Hydralisk Den`
`Zerg Infested Command Center`
`Zerg Lair`
`Zerg Nydus Canal`
`Zerg Queens Nest`
`Zerg Spawning Pool`
`Zerg Spire`
`Zerg Spore Colony`
`Zerg Sunken Colony`
`Zerg Ultralisk Cavern`