StarCraft EIS Environment Manual

Harm Griffioen, Danny Plenge, Vincent Koeman

10^{th} July, 2017

Contents

Chapter 1

Environment

This chapter will explain how to set up and start a bot with the StarCraft environment using a multi-agent system in the GOAL language.

1.1 Installation

For the latest installation instructions, please 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. You 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.

Alternatively, use http://www.openbw.com/replay-viewer to view replays in your browser.

1.3 Init Parameters

The StarCraft environment offers multiple configurable items through the init parametres of a mas2g file. When updating any parameters, do not forget to close the Chaoslauncher before launching a new game, as otherwise your changes will not be applied. The example below demonstrates all parameters and their defaults.

```
use "connector.jar" as environment with
    own_race="",
    enemy_race="random",
    map="",
    starcraft_location="C:\\StarCraft",
    auto_menu="SINGLE_PLAYER",
    game_type="MELEE",
    game_speed=50.
    debug="false",
    draw_mapinfo="false",
    draw_unitinfo="false",
    invulnernable="false",
    map_agent="true",
```

1.3.1 Own Race

You have to specify the race of your bot. This will make sure that the Chaoslauncher will automatically launch a game 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.

1.3.2 Enemy Race

The enemy race parameter can be used for specifying which race of the game's built-in AI you want to play against. To this end, you can insert *enemy_race=<RaceName>*, where *<RaceName>* can either be *zerg*, *protoss*, *terran*, *random*, *randomtp*, *randomtz*, or *randompz*. The option *random* will choose a race with a 1/3 chance for each race, whilst the other options will choose one of the two indicates races with a 1/2 chance for each race.

1.3.3 Map

You have 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*, and that subdirectories (like *sscait*) should be indicated. Also note that the first time the environment runs on a certain map, it will take some time (around 2 minutes) to generate a datafile for the given map (if not already present in *StarCraft/AI/BWTA*).

1.3.4 StarCraft Location

You have to specify the location of the StarCraft game if it is not installed in *C:*
Starcraft. Using this location, the Chaoslauncher will automatically start when launching a 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 (although you have to start the next game manually then). You can specify the location of StarCraft by inserting *StarCraft_location = <FilePath>*, where *<FilePath>* is the absolute path to the StarCraft installation folder.

1.3.5 Auto Menu

The auto menu parameter is used to automatically go through the menus of the game when starting a MAS. 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 Game Type

The game type is used to indicate what kind of game the Chaoslauncher should start. Generally, you want this to be the default (*MELEE*), but other game types can be used by inserting *game_type=<GameType>*.

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 (the speed can be changed during the game by using the development tool; see the next item). 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. 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>*. If a number lower than 1 is given, there will be no limit on the amount of FPS used, and the game will thus run as fast as it possibly can.

1.3.8 Debug

The environment offers a development tool for debugging purposes. With this development tool, you can increase or decrease the game speed, enable cheats and toggle the drawing of map and/or unit details in the game. More information about the development tool can be found in 1.5. In order to enable or disable launching the development tool, you can insert *debug=<Boolean>*.

1.3.9 Draw Map Info

This parameter can be used to draw info about the map (bases, regions, chokepoints) without having to enable it the development tool (or without starting the development tool at all) by inserting *draw_mapinfo=<Boolean>*.

1.3.10 Draw Unit Info

This parameter can be used to draw info about units (counts, IDs, health, targets) without having to enable it the development tool (or without starting the development tool at all) by inserting *draw_unitinfo=<Boolean>*.

1.3.11 Invulnerable

The invulnerable parameter can be used to automatically make your units invulnerable from the start of the game (which can also be done manually in the development tool). This can come in handy for testing purposes when you do not want to fight your opponent. To use the invulnerable function you can insert *invulnerable=<Boolean>*.

1.3.12 Map Agent

If enabled, the environment will generate an entity of name and type *'mapAgent'*. This entity is not connected to a unit in the game, but will be the only entity to receive all global information (*base/2, chokepoint/6, enemyrace/1, map/2, region/5, attacking/2, constructionSite/2-3, enemy/9, friendly/3, gameframe/1, mineralField/4, nuke/2, resources/4, underConstruction/5, vepseneGeyser/5, winner/1*), and can even take a few actions (*cancel/1, forfeit/0*). Thus, when the map agent is enabled, all other entities will only receive their own local information (*self/2, status/7, order/3, defensiveMatrix/1, queueSize/1, researching/1, unitLoaded/1*), whilst they can of course still take the same actions. The map agent entity can be enabled or disabled with *map_agent=<Boolean>*.

1.4 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. Please note that the environment will wait in the first game frame until *at least four actions* have been requested, e.g., until all initial workers have called *gather/1*. This will allow all initial agents (including the *'mapAgent'*) to fully start-up (and possible execute a few cycles already) before the game starts.

```
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 (an advanced Protoss ability), some units from other races can be taken over. These units will also get an entity. An possible way to accomodate such entities is by making sure any other unit type is

connected to some generic agent through a wildcard launch rule at the very end of your mas2g:

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

1.5 The Development Tool

The development tool can be automatically launched by using the *debug* init parameter. It provides several actions that are useful for debugging purposes.

1.5.1 Game Speed

The Game Speed slider can be found at the top of the development tool window. When the slider is used, the speed of the game will be changed immediately. The slider start on a value of 50 fps (this will not reflect the *game_speed* init parameter). The slowest speed is 20 fps, and from there you can set it as fast as you want. Please note that the agents are supposed to play at 50 fps, which is the default game speed for AI tournaments. When the speed is set to more than 100 fps, the agents can react slower than they would on the tournament speed. Setting the game speed to more than 100 fps should thus only be used for quick testing purposes.

1.5.2 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 damage* which makes the units of the player immune for damage (note: this can be automatically enabled with the init parameter *invulnerable*). 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.

1.5.3 Draw Actions

The development tool can also be used to show map or unit details in StarCraft itself. There are 2 buttons to this end, reflecting the matching *draw_mapinfo* and *draw_unitinfo* parameters. Please see the information above on these parameters for more information.

Chapter 2

Percepts

This chapter lists all the percepts that are generated by the StarCraft environment, which vary per unit (also see the *map_agent* init parameter in the previous chapter). For the implementation of these percepts in your agent program, we refer to the GOAL programming guide.

In order to reduce the number of percepts, one generic guideline used in this environment is to *only create percepts for information that changes in a single match or between matches.* Even though there is a lot of static information in a game like StarCraft, like the type of a unit (i.e., biological or mechanic), what a certain unit costs to produce, or the units a certain building can produce, this information remains the same for any execution of any agent system, and is thus much better suited to be encoded in the agent system itself. To this end, a Prolog file is supplied in the environment's installer that contains a large list of predicates representing static information about the game. The predicates available in this file are listed at the end of this section. Note that it is not expected that agents 'hard-code' information about specific maps on which matches can be played, and thus agents will need to be informed about changes between matches (i.e., map-specific information).

Another guideline used in this environment is that *no data is sent through percepts that can either be calculated based on other data, (e.g., the number of friendly units by adding the amount of percepts about their status) or retrieved from other agents (e.g., the position of a friendly unit).* Relaying such information through messaging(channels) is usually much more efficient, as one can then selectively choose at which times and to which units to send information, as opposed to percepts always being sent to certain units even when they do not require them (at that time) for their decision making.

2.1 Global Static Percepts

These percepts represent global information (i.e., not specific to a certain unit) that will not change during a match. When the map agent is enabled, it will be the only entity to receive these percepts when started. Otherwise, all entities will receive these percepts when started. Note that all coordinates (X,Y) reflect tile positions; one such tile is actually 16 by 16 pixels. Moreover, Zerg units that morph (or Protoss units that mege) into a different type of unit will keep the same ID (of one of the units).

2.1.1 base/4

Description: Information about all base locations on the map. These are possible construction sites for resource centers (or spawning sites for the initial center).

Type: Send once

Syntax: `base(<IsPossibleStart>,<X>,<Y>,<Region>)`

Example: `base(true, 28, 32, 8)`

Parameters:

`<IsPossibleStart>`	Indicates whether the location is a potential starting location or not (i.e., there are 2 on a 2-player map and 4 on a 4-player map).
Type	`Boolean`
Range	[true,false]
`<X>`	The x-coordinate of the base location.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the base location.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the base is located in. Can for example be used to find matching mineral patches and geysers.
Type	`Integer`
Range	[1–∞]

2.1.2 chokepoint/6

Description: Information about all 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 (depending on the specific chokepoint's width). All regions are connected through chokepoints.

Type: Send once

Syntax: `chokepoint(<X1>,<Y1>,<X2>,<Y2>,<Region1>,<Region2>)`

Example: `chokepoint(12, 15, 14, 17, 1, 2)`

Parameters:

<X1>	The x-coordinate of the first side.
Type	`Integer`
Range	[0–∞]
<Y1>	The y-coordinate of the first side.
Type	`Integer`
Range	[0–∞]
<X2>	The x-coordinate of the second side.
Type	`Integer`
Range	[0–∞]
<Y2>	The y-coordinate of the second side.
Type	`Integer`
Range	[0–∞]
<Region1>	The ID of the first region.
Type	`Integer`
Range	[1–∞]
<Region2>	The ID of the second region.
Type	`Integer`
Range	[1–∞]

2.1.3 enemyRace/1

Description	The race of the opponent (it is assumed there is only one).
Type	Send once
Syntax	`enemyRace(<Race>)`
Example	`enemyRace(protoss)`
Example	When playing against a random race (*<Race>=unknown*), you can use something like:

```
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.
Type	`String`
Range	*[terran,protoss,zerg,unknown]*

2.1.4 map/2

Description	The width and the height of the map (in tiles).
Type	Send once
Syntax	`map(<Width>,<Height>)`
Example	`map(96, 128)`

Parameters

<Width>	The width of the map (no. of horizontal tiles).
Type	`Integer`
Range	[1–∞]
<Height>	The height of the map (no. of vertical tiles).
Type	`Integer`
Range	[1–∞]

2.1.5 region/5

Description: Information about all regions on the map. Regions are connected by chokepoints 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	`Integer`
Range	[1–∞]
`<CenterX>`	The x-coordinate of the center of the region.
Type	`Integer`
Range	[0–∞]
`<CenterY>`	The y-coordinate of the center of the region.
Type	`Integer`
Range	[0–∞]
`<Height>`	The height of the region.
Type	`Integer`
Range	[0–∞]
`<ConnectedRegionsList>`	A list of regions (by ID) that are connected to this region (i.e., through chokepoints).
Type	`List`

2.2 Global Dynamic Percepts

These percepts represent information that changes during a match, but is still global to the match (i.e., not specific to a certain unit). When the map agent is enabled, it will be the only entity to receive these percepts during the match. Otherwise, all entities (or all workers for the *constructionSite/3-4* percept) will receive these percepts during the match.

2.2.1 attacking/2

Description: All enemy units that are attacking / going to attack and the corresponding units they have targeted (which can be friendly when e.g. healing or repairing).

Type: Send always

Syntax: `attacking(<Id>,<TargetId>)`

Example: `attacking(123, 177)`

Parameters:

`<Id>`	The ID of the enemy unit that is attacking.
Type	`Integer`
Range	[1–∞]
`<TargetId>`	The ID of the unit which is being targeted. This unit is mostly friendly, but can also be an enemy for e.g. heals or repairs.
Type	`Integer`
Range	[1–∞]

2.2.2 constructionSite/3-4

Description: All visible and non-obstructed locations at which buildings can potentially be constructed. Such construction sites are squares containing 4 tiles, as the minimum size of any building is 2 by 2 tiles. This information is updated every 50 game frames. Note that resource centers require a minimum distance to mineral patches and geysers (which the locations as indicated by *base/4* conform to for example).

Type: Send always

Syntax: (Protoss) `constructionSite(<X>,<Y>,<Region>,<InPylonRange>)`
(Zerg) `constructionSite(<X>,<Y>,<Region>,<OnCreep>)`
(Terran) `constructionSite(<X>,<Y>,<Region>)`

Example:
```
constructionSite(66, 98, 4, false)
constructionSite(66, 98, 4)
```

Parameters:

<X>	The x-coordinate of the construction site.
Type	`Integer`
Range	[0–∞]
<Y>	The y-coordinate of the construction site.
Type	`Integer`
Range	[0–∞]
<Region>	The region the construction site is in.
Type	`Integer`
Range	[1–∞]
<InPylonRange>	Indicates whether the construction site is in range of a pylon (for Protoss only).
Type	`Boolean`
Range	[True-False]
<OnCreep>	Indicates whether the construction site is on creep (for Zerg only).
Type	`Boolean`
Range	[True-False]

2.2.3 enemy/9

Description: Information about all visible (living) enemy units. Note that this also includes unfinished units (like buildings under construction) or cloaked units that cannot be attacked until they are detected (see *<Conditions>*). Such 'units' can also include spells: *'Spell Dark Swarm'*, *'Spell Distruption Web'*, and *'Spell Scanner Sweep'*.

Type: Send always

Syntax:
```
enemy(<Id>,<Type>,<Health>,<Shield>,<Energy>,
  <Conditions>,<X>,<Y>,<Region>)
```

Example:
```
enemy(12, 'Zerg Overlord', 200, 100, 0, [flying], 120, 96, 3)
```

Parameters

<Id>	The ID of the unit.
Type	Integer
Range	[1–∞]
<Type>	The type of the unit. This consists of a string with the race of the unit and the name of the unit parted by a space.
Type	String
Range	See Section 6
<Health>	The current amount of health of the unit.
Type	Integer
Range	[1–<maxHealth>] where <maxHealth> is the maximum health of the given unit type.
<Shield>	The current amount of shields of the unit.
Type	Integer
Range	[0–<maxShield>] where <maxShield> is the maximum shield of the given unit type.
<Energy>	The current amount of energy of the unit.
Type	Integer
Range	[0–<maxEnergy>] where <maxEnergy> is the maximum shield of the given unit type.
<Conditions>	A list representing the current conditions of the unit. Each unit can have multiple or no conditions depending on the unit and situation. Note that not all conditions are available for enemies.
Type	List of Strings
Range	See Section 2.5
<X>	The x-coordinate of the unit.
Type	Integer
Range	[0–∞]
<Y>	The y-coordinate of the unit.
Type	Integer
Range	[0–∞]
<Region>	The region the unit is in. Can be 0 if the unit is on a chokepoint (and thus 'in-between' regions).
Type	Integer
Range	[0–∞]

2.2.4 friendly/2

Description: Information about all (living) units of the player. Note that this also includes unfinished units that do not have an entity and thus agent yet (like buildings under construction), see also the *underConstruction/5* percept.

Type: Send always

Syntax: `friendly(<Id>,<Type>)`

Example: `friendly(26, 'Protoss Gateway')`

Parameters:

<Id>	The ID of the unit.
Type	`Integer`
Range	[1–∞]
<Type>	The type of the unit. This consists of a string with the race of the unit and the name of the unit parted by a space.
Type	`String`
Range	See Section 6

2.2.5 gameframe/1

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: `gameframe(<Number>)`

Example: `gameframe(150)`

Parameters:

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

2.2.6 mineralField/5

Description: Information about visible (non-empty) mineral fields.
Type: Send always
Syntax: `mineralField(<Id>,<Resources>,<X>,<Y>,<Region>)`
Example: `mineralField(57, 5000, 6, 22, 32)`

Parameters

<Id>	The ID of the mineral field.
Type	Integer
Range	[1–∞]
<Resources>	The amount of minerals left in the field. Can be 0 for mineral fields that obstruct e.g. chokepoints.
Type	Integer
Range	[0–5000]
<X>	The x-coordinate of the mineral field.
Type	Integer
Range	[0–∞]
<Y>	The y-coordinate of the mineral field.
Type	Integer
Range	[0–∞]
<Region>	The region the mineral field is in.
Type	Integer
Range	[1–∞]

2.2.7 nuke/2

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

Parameters

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

2.2.8 resources/4

Description: The amount of minerals, gas and supply available to the player (i.e. shared by all units). Note that in order to avoid halves, 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.
Type	`Integer`
Range	[0–∞]
`<G>`	The current amount of gas available.
Type	`Integer`
Range	[0–∞]
`<CS>`	The amount of supply that is in use.
Type	`Integer`
Range	[0–400]
`<TS>`	The total amount of supply that is available. Note that usually `<TS>` is always greater or equal to `<CS>`, but this can change when supply providing units are killed.
Type	`Integer`
Range	[0–400]

2.2.9 underConstruction/5

Description: Indicates a new friendly unit that is under construction (i.e. by a worker or through morphing). This percept allows getting data about unfinished units that do not have an entity (and thus agent) yet.

Type: Send always

Syntax: `underConstruction(<Id>,<Vitality>,<X>,<Y>,<Region>)`

Example: `underConstruction(44, 74, 22, 37, 2)`

Parameters:

`<Id>`	The ID of the unit.
Type	`Integer`
Range	[1–∞]
`<Vitality>`	The combined amount of health and shield of the unit. This generally increases whilst the unit is nearing completion, though it can simultaneously be brought done as well by enemy attacks.
Type	`Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the unit is in. Can be 0 if a morphing unit is on a chokepoint (and thus 'in-between' regions).
Type	`Integer`
Range	[0–∞]

2.2.10 vespeneGeyser/5

Description: Information about visible (though possibly empty) vespene geysers. Empty geysers can still be mined from, though at a reduced rate.
Type: Send always
Syntax: `vespeneGeyser(<Id>,<Resources>,<X>,<Y>,<Region>)`
Example: `vespeneGeyser(57, 5000, 22, 32, 6)`
Parameters:

`<Id>`	The ID of the vespene geyser.
Type	`Integer`
Range	[1–∞]
`<Resources>`	The amount of gas left in the vespene geyser.
Type	`Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the vespene geyser.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the vespene geyser.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the vespene geyser is in.
Type	`Integer`
Range	[1–∞]

2.2.11 winner/1

Description: Indicates if the player has won or lost at the end of the game. Used mainly for automated testing purposes.
Type: Send once
Syntax: `winner(<HasWon>)`
Example: `winner(true)`
Parameters:

`<HasWon>`	Whether the player has won or not.
Type	`Boolean`
Range	[true,false]

2.3 Generic Unit Percepts

These percepts are generated for all individual units in the game (thus excluding the mapAgent if it is enabled). Although the *self/2* percept represents static information that does not change during the match, the *status/7* and the *order/3* percepts are updated frequently.

2.3.1 self/2

Description	Indicates the ID and type of the unit itself.
Type	Send once
Syntax	`self(<ID>, <Type>)`
Example	`self(21, 'Terran SCV')`

Parameters

<Id>	The ID of the unit.
Type	`Integer`
Range	[1–∞]
<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`

2.3.2 status/7

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>, <Conditions>,
  <X>, <Y>, <Region>)
```

Example:
```
status(250, 0, 0, [moving, carrying], 24, 36, 1)
```

Parameters

`<Health>`	The current amount of health of the unit.
Type	`Integer`
Range	[0–`<MaxHealth>`] where `<MaxHealth>` is the maximum health of the given unit.
`<Shield>`	The current amount of shields of the unit.
Type	`Integer`
Range	[0–`<MaxShield>`] where `<MaxShield>` is the maximum shield of the given unit.
`<Energy>`	The current amount of energy of the unit.
Type	`Integer`
Range	[0–`<MaxEnergy>`] where `<MaxEnergy>` is the maximum energy of the given unit.
`<Conditions>`	A list representing the current conditions of the unit. Each unit can have multiple or no conditions depending on the unit and situation.
Type	`List of Strings`
Range	See Section 2.5
`<X>`	The x-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the unit is in. Can be 0 if the unit is on a chokepoint (and thus 'in-between' regions).
Type	`Integer`
Range	[0–∞]

2.3.3 order/5

Description: Indicates what the unit is ordered to do. A unit always has an order (e.g. even 'Nothing' is an order).

Type: Send on change

Syntax: `order(<Primary>, <TargetUnit>, <TargetX>, <TargetY>, <Secondary>)`

Example: `order('AttackMove', -1, 34, 8, 'None')`

Parameters:

`<Primary>`	The primary order of the unit. Some actions are converted into race or unit specific orders.
Type	`String`
Range	See https://bwapi.github.io/namespace_b_w_a_p_i_1_1_orders.html
`<TargetUnit>`	The ID of the unit the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
`<TargetX>`	The X coordinate of the position the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
`<TargetY>`	The Y coordinate of the position the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
`<Secondary>`	The secondary order of the unit. This is usually 'None', but is used when for example a Protoss Carrier is both moving and producing units.
Type	`String`
Range	See https://bwapi.github.io/namespace_b_w_a_p_i_1_1_orders.html

2.4 Unit-Specific Percepts

These percepts are generated only for specific units in the game. All of these (dynamic) percepts represent information that can change during the match.

2.4.1 defensiveMatrix/1

Description	Information about how much health the defensive matrix has left on the unit. This only applies to friendly Terran units having received such a matrix from a Science Vessel.
Type	Send always
Syntax	`defensiveMatrix(<health>)`
Example	`defensiveMatrix(200)`

Parameters

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

2.4.2 queueSize/1

Description	The number of units that are in the queue of any production unit (e.g. including Protoss Reavers and Carriers). Zerg Hatchery/Lair/Hive: The number of of available larva. Terran Nuclear Silo: 1 if a nuke is ready (after a corresponding *train/1* action); 0 otherwise.
Type	Send on change
Syntax	`queueSize(<Size>)`
Example	`queueSize(2)`

Parameters

`<Size>`	See description.
Type	`Integer`
Range	[0–5]

2.4.3 researching/1

Description	Indicates which technology or upgrade is being researched by the unit (i.e. a building).
Type	Send always
Syntax	`researching(<Type>)`
Example	`researching('Stim Packs')`

Parameters

<Type>	The technology or upgrade that is being researched.
Type	`String`
Range	See Section 4 and Section 5

2.4.4 unitLoaded/1

Description	Indicates which unit(s) are loaded inside the loadable unit (e.g. a Terran Bunker or a Protoss Shuttle).
Type	Send always
Syntax	`unitLoaded(<Id>)`
Example	`unitLoaded(154)`

Parameters

<Id>	The ID of the loaded unit.
Type	`Integer`
Range	[0–∞]

2.5 Conditions

The conditions a unit can potentially have. The race-specific conditions are either only applicable to or caused by units from that race.

2.5.1 Workers

`carrying`	Indicates when the worker unit is carrying minerals or vespene gas.
`constructing`	Shows that the worker unit is busy constructing a building.
`gathering`	Show that the worker unit is busy gathering minerals or vespene gas.
`repairing`	Shows that the (Terran) worker unit is busy repairing a building.

2.5.2 Generic

`attacking`	Indicates when a unit is attacking an other unit (includes medic heal).
`beingConstructed`	Indicates that a unit is incomplete (includes morphing).
`cloaked`	Indicates that a unit is cloaked.
`coolingDown`	Indicates that a unit cannot attack due to cooldown.
`detected`	Indicates that an enemy cloaked/burrowed unit has been detected (and thus can be attacked).
`flying`	Indicates that a unit is flying.
`following`	Indicates that a unit is following an other unit.
`holding`	Indicates that a unit is holding a position.
`idle`	Indicates that the unit is idle (not doing anything).
`loaded`	Indicates that a unit is loaded (i.e. has one or more units in it).
`moving`	Indicates that a unit is moving.
`patrolling`	Indicates that a unit is patrolling between 2 positions.
`underAttack`	Indicates that a unit is under attack.

2.5.3 Zerg

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

2.5.4 Terran

`<addonName>`	Indicates that an addon of a Terran building is present.
`beingHealed`	Indicates that a Terran unit is being healed by a Medic or repaired by a SCV.
`blinded`	Indicates that a unit is blinded by a Terran Medic.
`defenseMatrixed`	Indicates that a Terran unit has a defensive matrix on it (from a Science Vessel).
`hasMines`	Indicates that a Terran Vulture has at least one Spider Mine.
`irradiated`	Indicates that a unit is irradiated by a Terran Science Vessel.
`lifted`	Indicates that a Terran building is lifted (and thus can move).
`lockDowned`	Indicates that a unit is under lockdown by Terran Ghost.
`nukeReady`	Indicates that a Terran Nuclear Silo is ready to launch a Nuclear Missile.
`sieged`	Indicates that a Terran Siegetank is in siegemode.
`stimmed`	Indicates that a Terran Firebat or Marine is stimmed.

2.5.5 Protoss

`disruptionWebbed`	Indicates that a unit is in a disruption web from a Protoss Corsair.
`hasScarabs`	Indicates that a Protoss Reaver has at least one Scarab.
`maelstrommed`	Indicates that a unit is maelstrommed by a Protoss Dark Archon.
`stasised`	Indicates that a unit is stuck in stasis from a Protoss Arbiter.
`underStorm`	Indicates that a unit is under a storm from a Protoss High Templar.
`unpowered`	Indicates that a Protoss building unit is no longer powered by a pylon (e.g. a Photon Cannon then no longer functions).

Chapter 3

Actions

This chapter lists all the actions that are usable in the StarCraft environment, which vary per unit.

3.1 All Units

The following actions can be executed by any unit.

3.1.1 cancel/1

Description	Cancel the construction or morphing of a unit.
Syntax	`cancel(<TargetId>)`
Parameters	`<TargetId>`: The ID of the unit of which the construction or morphing should be cancelled.
Pre	The targeted unit is incomplete (not fully constructed or morphed).
Post	The targeted unit's construction or morphing will be cancelled; resources will be refunded (and for Zerg the original unit will be restored).
Note	It makes most sense for the *'mapAgent'* to execute this.

3.1.2 debugdraw/1

Description	Draw text above a unit in the game window.
Syntax	`debugdraw(<Text>)`
Parameters	`<Text>`: The text(string) that should be drawn.
Pre	-
Post	The given text will be drawn above the unit (i.e., it will stay with the unit) in the game window. If the given text is empty, the drawing will be cancelled.
Note	For the *'mapAgent'*, the text will be drawn on a fixed position on the left top of the game window.

3.1.3 forfeit/0

Description	Forfeit the game.
Syntax	`forfeit`
Pre	The game is in progress.
Post	The game ends with a loss for the player.
Note	It makes most sense for the *'mapAgent'* to execute this.

3.1.4 morph/1 (Zerg only)

Description	Morph a unit into another unit(type).
Syntax	`morph(<Type>)`
Parameters	`<Type>`: The type to morph into. See 6.
Pre	The unit is capable of morphing into the given unit type.
Post	The unit's corresponding agent terminates and a new agent is created for the new unit when it is completed (with the same ID).

3.2 Buildings

The actions in this section can only be executed by buildings (or by some special units that can be loaded or that can produce units of their own).

3.2.1 buildAddon/1 (Terran only)

Description	Build an addon.
Syntax	`buildAddon(<Name>)`
Parameters	`<Name>`: The name of the addon that is to be constructed. See 6.
Pre	The building is capable of building the addon and does not already have an addon.
Post	The building starts constructing the addon.

3.2.2 cancel/0

Desription	Cancel the last train or research action.
Syntax	`cancel`
Pre	The unit is training, researching, or constructing an add-on (Terran-only).
Post	The last train, research, or add-on build is cancelled; the resources are refunded.

3.2.3 land/2 (Terran only)

Desription	Land a lifted building on a given location.
Syntax	`land(<X>, <Y>)`
Parameters	`<X>`: The x-coordinate of the chosen landing location. `<Y>`: The y-coordinate of the chosen landing location.
Pre	The unit is currently lifted and the landing location is visible, not obstructed, and fitting for the building.
Post	The unit moves to (if needed) and lands on the chosen location. It reconnects with any addon if applicable.

3.2.4 lift/0 (Terran only)

Description	Lift a building into the air.
Syntax	`lift`
Pre	The unit is capable of lifting and is not currently performing any other action.
Post	The building lifts into the air.

3.2.5 load/2

Description	Load a given unit into the unit.
Syntax	`load(<Id>)`
Parameters	`<Id>`: The ID of the unit to load into this unit.
Pre	The unit is capable of loading the targeted unit and has enough space provided for the targeted unit.
Post	The targeted unit moves towards to the loadable unit and loads into it.

3.2.6 research/1

Description	Research a tech or upgrade.
Syntax	`research(<Type>)`
Parameters	`<Type>`: The name of the tech or upgrade. See 5 and 4.
Pre	The unit is capable of researching the given tech or upgrade.
Post	The unit starts researching the given tech or upgrade.

3.2.7 train/1

Description	Train a unit.
Syntax	`train(<Type>)`
Parameters	`<Type>`: The type of unit to train. See 6.
Pre	The unit is capable of producing the given unit.
Post	The unit starts producing the given unit.

3.2.8 unload/1

Description	Unload a loaded unit from the unit.
Syntax	`unload(<Id>)`
Parameters	`<Id>`: The ID of the unit to unload from this unit.
Pre	The given unit is currently loaded into the unit.
Post	The targeted unit is unloaded from the unit.

3.2.9 unloadAll/0

Description	Unload all loaded units from the unit.
Syntax	`unloadAll`
Pre	There are units currently loaded into the unit.
Post	All loaded units are unloaded from the unit.

3.3 Moving Units

The action in this section can only be executed by moving units (i.e. non-buildings or lifted Terran buildings).

3.3.1 ability/1

Description	Use a (researched) ability.
Syntax	`ability(<Type>)`
Parameters	`<Type>`: The type of technology to use. See 4.
Pre	The given TechType is researched and the unit is capable of performing the ability (without a target unit or location).
Post	The unit performs the ability.
Note	Behaviour that can be toggled on and off (e.g. Burrow/-Cloak/Siege) is also executed by using this action (i.e. once for enabling and then again for disabling).

3.3.2 ability/2

Description	Use a (researched) ability on a target unit.
Syntax	`ability(<Type>, <Target>)`
Parameters	`<Type>`: The type of technology to use. See 4. `<Target>`: The target to use the technology on.
Pre	The given TechType is researched, the unit is capable of performing the ability (with some target unit), and the target unit is visible
Post	The unit performs the ability on the target unit.

3.3.3 ability/3

Description	Use a (researched) ability on a location.
Syntax	`ability(<Type>, <X>, <Y>)`
Parameters	`<Type>`: The type of technology to use. See 4. `<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	The chosen TechType is researched, the unit is capable of performing the ability (with some target location), and the location is visible.
Post	The unit performs the ability on the given location.

3.3.4 attack/1

Description	Attack a given unit.
Syntax	`attack(<TargetId>)`
Parameters	`<TargetId>`: The ID of the unit that should be attacked.
Pre	The unit is attack capable and the targeted unit is visible and reachable.
Post	The targeted unit is being attacked by your unit. The unit will keep moving towards the enemy unit in order to attack it as long as it is visible and alive.
Note	Terran Medics can use this action to heal friendly units; they cannot attack enemies.

3.3.5 attack/2

Description	Move to a given location and attack everything on the way.
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 attack capable.
Post	The unit moves to the chosen location (or as close as it can get) whilst attacking any enemy unit that it encounters along the way; all such enemy units will be chased until they are no longer visible or alive.
Note	Terran Medics will heal any friendly units they encounter.

3.3.6 follow/1

Description	Follow a given unit.
Syntax	`follow(<given>)`
Parameters	`<given>`: The ID of the unit that should be followed.
Pre	The targeted unit is visible.
Post	The unit follows the selected unit; any enemy will be ignored (i.e. the unit will not automatically attack anything).

3.3.7 hold/0

Description	Hold a position.
Syntax	`hold`
Pre	-
Post	The unit will hold its current position; any enemy will be ignored (i.e. the unit will not automatically attack anything).

3.3.8 move/2

Description	Move to a given location.
Syntax	`move(<X>,<Y>)`
Parameters	`<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	-
Post	The unit moves to the chosen location (or as close as it can get) whilst ignoring any enemy unit along the way (i.e. the unit will not automatically attack anything).

3.3.9 patrol/2

Description	Patrol between a unit's current position 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	-
Post	The unit patrols between its current position and the chosen location (or as close as it can get); any enemy unit that it encounters will be chased until it is no longer visible or alive, after which the unit will return to its patrol route.
Note	Terran Medics will heal any friendly units they encounter.

3.3.10 stop/0

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

3.4 Workers

The actions in this section can only be executed by worker units.

3.4.1 build/3

Description	Build a building on the given location.
Syntax	`build(<Type>,<X>,<Y>)`
Parameters	`<Type>`: The type of building that should be built. See 6. `<X>`: The x-coordinate of the build location `<Y>`: The y-coordinate of the build location
Pre	The unit is capable of constructing the chosen building and the build location is visible, not obstructed, and fitting for the given building.
Post	The unit goes moves the build location (if needed) and starts constructing the building there. Zerg Drones will morph (i.e., the drone will be lost), Terran SCVs will be busy constructing for a while, and Protoss Probes will instantiate a warp (i.e., the probe does not have to remain at the build location). See also *cancel/1* and *repair/1*.

3.4.2 gather/1

Description	Gather a specific resource (minerals or vespene gas building).
Syntax	`gather(<Id>)`
Parameters	`<Id>`: The ID of the chosen resource.
Pre	The given resource is visible and reachable.
Post	The unit starts gathering the chosen resource. It automatically moves back and forth between the resource and the closest resource center.

3.4.3 repair/1 (Terran only)

Description	Repair a unit or complete an unfinished building.
Syntax	`repair(<Id>)`
Parameters	`<Id>`: The ID of the unit to repair or of the building to complete construction of.
Pre	The unit is a Terran SCV, has the resources to repair, and the target unit is visible (and reachable)
Post	The SCV moves towards the selected unit (if needed) and repairs it or resumes its construction.

Chapter 4

Tech Types

All tech types that can be researched for each race and can thus be used as abilities by the indicated units.

4.1 Terran

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

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

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 upgrade types that can be researched for each race at the indicated buildings.

5.1 Terran

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

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

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

Zerg Flyer Carapace
Zerg Flyer Attacks

5.3.8 Ultralisk Cavern

Chitinous Plating
Anabolic Synthesis

Chapter 6

Unit Types

StarCraft's unit types.

6.1 Terran

6.1.1 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 Air Units

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

6.1.3 Buildings

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

6.1.4 Addons

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

6.2 Protoss

6.2.1 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 Air Units

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

6.2.3 Buildings

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

6.3.1 Ground Units

Zerg Broodling
Zerg Defiler
Zerg Drone
Zerg Egg
Zerg Hydralisk
Zerg Larva
Zerg Lurker

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

6.3.2 Air Units

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

6.3.3 Buildings

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