A.I. Command Language:

Quick Reference

; [comments]
assign v# ~v#
assign v# [any value]
assign v# #[system_variable]
attempt repairs
author [developer's name]
beep
cross scan
corner scan
debug on
debug off
discharge energy
fire weapon
generate random
gosub [routine name]
goto [line label]
gps scan x [x position] **y** [y position]
if ammo is > # then [additional command]
if ammo is < # then [additional command]
if ammo is = # then [additional command]
if bump barrier then [additional command]
if damage is > # then [additional command]
if damage is < # then [additional command]
if damage is = # then [additional command]
if facing north then [additional command]
if facing south then [additional command]
if facing east then [additional command]
if facing west then [additional command]
if fuel is > # then [additional command]
if fuel is < # then [additional command]
if fuel is = # then [additional command]
if missile ready then [additional command]
if no ammo then [additional command]
if random is 1 then [additional command]
if random is 2 then [additional command]
if random is 3 then [additional command]
if random is 4 then [additional command]
if scan found barrier then [additional command]
if scan found enemy then [additional command]
if scan found friend then [additional command]
if scan found flag then [additional command]
if scan found mine then [additional command]
if scan found nothing then [additional command]
if shield is up then [additional command]
if shield is down then [additional command]
if...then...end if
if x coordinate is < # then [additional command]
if x coordinate is > # then [additional command]
if x coordinate is = # then [additional command]
if y coordinate is < # then [additional command]
if y coordinate is > # then [additional command]
if y coordinate is = # then [additional command]
if value (variable) **(**>, <, =, <>, >=, <=**)** (value or variable**) then** [additional command]
iff code [any code number or word(s)]
launch missile

lay mines on
lay mines off
long range scan
lower shield
math v# = (value) (+, -, *, /) (value) **…**
move forward
move backward
name [unit name]
password [security password]
raise shield
return
scan forward
scan right
scan left
scan perimeter
scan position 1
scan position 2
scan position 3
scan position 4
scan position 5
scan position 6
scan position 7
scan position 8
self destruct
turn right
turn left

Variables:
v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, va, vb, vc, vd, ve, vf, vg, vh, vi, vj, vk, vl, vm ,vn ,vo ,vp, vq, vr, vs, vt, vu, vv, vw, vx ,vy, vz
#cur_fuel
#max_fuel
#cur_ammo
#set_ammo
#cur_score
#random
#scan
#shield
#burn
#x_pos
#y_pos
#cur_life
#max_life
#cur_head
#enemy_x
#enemy_y
#enemy_h
#enemy_d
#friend_x
#friend_y
#friend_h
#friend_d

Command Language Syntax

All commands must be typed using the syntax given including spaces.

The A.I. compiler converts most commands to lower case before interpreting.

Unit Identification and Security

name [unit name] - This sets the units name.
Example:

```
name Warrior 1
```

author [player name] - This sets the authors name.
Example:

```
name John Doe
```

password [security password] - This sets the units security password. If this line is entered in the units code the units AI will not allow editing or debugging without a proper security password entered in the battle setup screen.
Example:

```
password allison
```

iff code [any code number or word(s)] - Identify friend or foe. Units with matching iff codes will show up as a friend on any scan.
Example:

```
iff code read team
```

Debugging

debug on - saves any following commands that are seen by the unit to the debug watch buffer. This buffer can be viewed by selecting the **debug watch** button from the battle summary screen.
Example:

```
debug on
```

WARNING: you may wish to remove these statements from AI files that you distribute because this will allow other users to see portions of your code when they view the debug watch for that unit.

Note: an active debug watch will slow down the battle due to the extra processing overhead created.

debug off - stops saving command information to the **debug watch** buffer.
Example:

```
debug off
```

Debug on and off are used with the **debug watch** screen. This screen is only available in the registered version.

beep - Plays the windows default beep sound. Useful in debug mode to determine if a section of code is being read in the units AI.
Example:

```
if bump barrier then beep
```

Note:

; Comments: Lines that begin with a semi-colon are considered comments and will be ignored by the AI interpreter.
Example:

```
; this is a comment line
```

Program Branching

goto [label name] - This will jump from one section of the units AI code to another line (label) in the program. Label names can be anything followed by a colon ":".
Example:

```
start:
move forward
goto start
```

gosub [routine name] - This will jump you from one section of the units AI code to another sub routine. Sub routines start with line labels and end with the keyword **return**.
Example:

```
start:
move forward
if bump barrier then gosub bhit
goto start

bhit:
generate random
if random is 1 then turn right
if random is 2 then turn left
if random is 3 then gosub turnit
if random is 4 then fire weapon
return

turnit:
turn right
turn right
return
```

Unit Damage Status and Repairs

if damage is > # then [additional command]
if damage is < # then [additional command]
if damage is = # then [additional command]
If damage is greater than, less than, or equal to the percent given then execute the command to the right of the word then.
Note: do not include a percent sigh "%". Valid percent entries are 0 through 99
Example:

```
if damage is > 95 then self destruct
```

if fuel is > # then [additional command]
if fuel is < # then [additional command]
if fuel is = # then [additional command]
If fuel is greater than, less than, or equal to the percent given then execute the command to the right of the word then.
Note: do not include a percent sigh "%". Valid percent entries are 0 through 99
Example:

```
if fuel is < 100 then goto hide
```

attempt repairs - This will try to repair any damage that the unit has. You have a 1 in 10 chance of it working. This requires a lot of energy therefore you cannot have shields raised while attempting this.

WARNING: There is also a 5 Percent chance that you will damage the fuel system when you attempt repairs during operation this will cause your unit to burn fuel less efficiently.

Advanced commands:
#cur_life & **#max_life**: see system variables
#cur_fuel & **#max_fuel**: See system variables.
#burn: See system variables.

Randomizing Units AI

generate random - generates a random number between 1 and 4.
Example:

```
generate random
```

if random is 1 then [additional command]
if random is 2 then [additional command]
if random is 3 then [additional command]
if random is 4 then [additional command]
If the random number generated by the generate random command is equal to the number specified then execute the command to the right of the word then
Example:

```
if random is 4 then turn right
```

Advanced command:
#random: see system variables

Scanning for objects and other units

long range scan - This will scan the entire distance between the unit and the edge of the battlefield.

gps scan x [x position] **y** [y position] - global position scanner looks at the specified x and y coordinates and returns what it finds. The x position and y position are the x and y coordinates of the battle map.
Examples:

```
gps scan x 10 y 13
if scan found enemy then
      gosub hunter
end if
```

or

```
gps scan x #enemy_x y #enemy_y
```

Coordinates (top left corner = x:1, y:1, bottom right corner x:43, y:30)

scan forward - This will scan the area of 5 spaces out directly in front of the unit.

Note: 5 spaces is the range of the primary weapon.

scan right - This will scan the area of 5 spaces out from the right of the unit.

scan left - This will scan the area of 5 spaces out from the left of the unit.

scan perimeter - This will scan all spaces directly around the unit.

scan position 1 - This scans one space directly north of the unit.

scan position 2 - This scans one space directly northeast of the unit.

scan position 3 -This scans one space directly east of the unit.

scan position 4 -This scans one space directly southeast of the unit.

scan position 5 -This scans one space directly south of the unit.

scan position 6 -This scans one space directly southwest of the unit.

scan position 7 -This scans one space directly west of the unit

scan position 8 -This scans one space directly northwest of the unit.

cross scan - This scans one space north, south, east and west of the unit.

corner scan - This scans one space northeast, southeast, northwest and southwest of the unit.

if scan found barrier then [additional command]
if scan found enemy then [additional command]
if scan found friend then [additional command]
if scan found flag then [additional command]
if scan found mine then [additional command]
if scan found nothing then [additional command]
If the scan returns the presence of a a barrier, enemy, friend or flag then execute the command to the right of the word then.
Note: If a scan that searches in more than one direction such as the cross scan, perimeter and corner scan finds more than one object type (i.e.; barrier flag and enemy) the scan will return items in the following priority (the top takes priority over the bottom):

- Enemy / Friend
- Mine
- Flag
- Barrier

Example:

```
scan position 5
if scan found enemy fire weapon
```

Advanced commands:
#scan, **#enemy_h** and **#enemy_d** : see system variables.

#enemy_x & **#enemy_y**

To help your unit locate other units in battle, you may use the variables **#enemy_x** and **#enemy_y**. These variables will give you the X and Y locations of the closest AI unit (friend or enemy). See the System variables section for more information on how to use these variables. The Tracker.ai program is an example of how to utilize these variables.

Example:

```
if x coordinate is = #enemy_x then turn
right
```

Movement and location

move forward - Moves unit one space forward in its current direction. This command will have no effect if a barrier blocks its path.
Example:

```
move forward
```

move backward - Moves unit one space backwards from its current direction. This command will have no effect if a barrier blocks its path.
Example:

```
move backward
```

turn right - Turns unit to face right from its current direction.
Example:

```
turn right
```

turn left - Turns unit to face left from its current direction.
Example

```
turn left
```

if facing north then [additional command]
if facing south then [additional command]
if facing east then [additional command]
if facing west then [additional command]
If the unit is facing north, south, east or west then execute the command to the right of the word then.
Example:

```
if facing east then turn left
```

if x coordinate is < # then [additional command]
if x coordinate is > # then [additional command]
if x coordinate is = # then [additional command]
if y coordinate is < # then [additional command]
if y coordinate is > # then [additional command]
if y coordinate is = # then [additional command]
If the units X or Y coordinate equals the number specified then execute the command to the right of the word then.
Coordinates (top left corner = x:1, y:1, bottom right corner x:43, y:30)

Example:

```
if x coordinate = 18 then turn right
```

if bump barrier then [additional command] - Use this to check and see if movement is blocked by a barrier. If a barrier is blocking the units path then it will execute the command to the right of the word then.
Example:

```
if bump barrier then turn right
```

Advanced Command:

#cur_head	closest enemy heading 1=North 2=East 3=South 4=West

Weapons Control

fire weapon - Fires the units primary weapon. This is a projectile that does maximum damage to an unshielded enemy unit at close range. Shields and range effect the amount of damage given. Shielded units are completely protected from medium and long range shots.
Note: the range of the weapon is 5 spaces. You cannot fire this weapon when shields are raised.
Example:

```
fire weapon
```

launch missile - Launches missile. The missile will travel in the direction fired until a object is hit. Missiles do 90% damage to units that are hit with their shields down. They do 70% damage to any unit nearby the detonation with their shields down. Units with shields up will receive 30% less damage overall.
WARNING: do not fire the missile with your shields up. The missile will misfire doing 90% damage to the launching unit.

Note: Missiles require 300 fuel and 10 ammo to fire.
Example:

```
if ammo is > 10 then
      launch missile
end if
```

discharge energy - This will discharge a blast of energy from your unit causing it one point of damage. Any enemy unit caught in this blast will take 3 points of damage. A energy discharge will destroy any flags and mines in the blast area. This is a good way to sweep for mines and deny any other players flags if your damage is zero.
Example:

```
if scan found enemy then
      discharge energy
end if
```

self destruct - This is a last resort. A unit that self destructs will not leave a flag. Any unit caught in the blast wave of a self destructing unit will receive blast wave damage equal to the amount of damage points remaining on the destructing unit and the maximum damage setting. Example: if maximum damage is set to 10 and the destructing unit has 3 damage points then the blast wave will do 7 points of damage to any unit directly next to the self destructing unit.
Example:

```
if damage is > 95 then self destruct
```

lay mines on - While this is on the unit will lay a mine every time it moves forward or backward one space. A mine requires 2 ammo to produce.
Example:

```
if scan found enemy then
      lay mines on
end if
```

lay mines off - This stops the laying of mines when the unit moves one space forward or backward. A mine requires 2 ammo to produce.

if ammo is > # then [additional command]
if ammo is < # then [additional command]
if ammo is = # then [additional command]
If the units ammo is greater than, less than or equal to the number given then execute the command to the right of the word then.
Note the maximum amount of ammo a unit can carry is 99.
Example:

```
if ammo is > 10 then lay mines on
```
or
```
if ammo is < 10 then lay mines off
```

if no ammo then [additional command]
If the unit has no ammo remaining then execute the command to the right of the word then.
Example:

```
if no ammo then goto hideout
```

if missile ready then [additional command]
If the missile has enough fuel and ammo to launch then do the command to the right of the word then.
Example:

```
if missile ready then launch missile
```

Note: this doesn't check the status of the shield be sure to lower shield before firing a missile.

Advanced Commands:
#cur_ammo & **#set_ammo**: See system variables.

Protection

raise shield - Raises shield to protect unit from medium and long range enemy fire and minimizes short range weapon blasts. The shield only protects against projectiles the shield is defenseless against energy discharges , overloads and self destruct blast waves. The shield requires most of the units power therefore, you cannot fire weapon or attempt repairs with the shield raised.
Example:

```
lower shield
fire weapon
raise shield
```

lower shield - Lowers shield to allow weapon firing and to attempt repairs.

if shield is up then [additional command]
if shield is down then [additional command]
If the units shield is raises or lowered then execute the command to the right of the word then.
Example:

```
if shield is down then fire weapon
```

Advanced Command:
#shield: See system variables.

Advanced Commands

Nesting:

if … then
…
end if

You may nest if statements by using the following syntax:

```
if scan found flag then
      if damage is = 0 then
            turn right
            turn right
      end if
end if
```

the command compiler sees these commands in this manner:

line 1:
```
if scan found flag then if damage is  = 0
then turn right
```

line 2:
```
if scan found flag then if damage is  = 0
then turn right
```

Warning: you must be careful not to check for two conditions at the same time for example:

```
if scan found flag then
      scan forward
      if scan found barrier then
            turn right
      end if
end if
```

the command compiler sees these commands in this manner:

line 1:
```
if scan found flag then scan forward
```

line 2:
```
if scan found flag then if scan found
barrier then turn right
```

Notice that line 2 cannot work because scan cannot be both a flag and a barrier.

Keep in mind how the compiler sees nested commands so you do not fall into this trap.

User Variables:

You have 36 user variables that you can use in your AI code:

v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, va, vb, vc, vd, ve, vf, vg, vh, vi, vj, vk, vl, vm ,vn ,vo ,vp, vq, vr, vs, vt, vu, vv, vw, vx ,vy and vz

you may assign these variables any value and or text you wish and use them throughout your AI code.

Note: remember that v0 and vo are not the same variable one uses the Number zero and the other uses the letter “o”.

variables have their names and their values:

if you want to reference its value you must place a tilde in front of it’s name example:

if the variable v2 had a value of 67 and you wanted to use its value in a command you would address v2 in the following manner:

```
assign v7 ~v2
```

This command passes the value of 67 to the variable v7.

These variables can also be manipulated using the following commands and system variables:

System Variables

System variables can be referenced but cannot be changed. They always begin with the "#" symbol.

The system variables are:

#cur_fuel	Current fuel value
#max_fuel	Battle Start fuel setting
#cur_ammo	Current ammo value
#set_ammo	Battle start ammo setting
#cur_score	Current score
#random	Last random number generated
#scan	Last scanned item 0 = nothing 1 = barrier 2 = enemy 3 = mine 4 = friend 5 = flag
#shield	shield status 1 = on, 0 = off.
#burn	current fuel burn rate Note: The burn rate will increase every time the unit sustains damage.
#x_pos	current unit X coordinate
#y_pos	current unit Y coordinate
#cur_life	current damage value
#max_life	maximum damage setting
#cur_head	current heading 1=North 2=East 3=South 4=West
#enemy_x	closest enemy x position
#enemy_y	closest enemy y position
#enemy_h	closest enemy heading 1=North 2=East 3=South 4=West
#enemy_d	closest enemy damage value
#friend_x	closest friend x position
#friend_y	closest friend y position

#friend_h	closest friend heading 1=North 2=East 3=South 4=West
#friend_d	closest friend damage value

Advanced Commands:

assign v# ~v#
assign v# [any value**]**
assign v# #(system_variable**)**

The assign command assigns a variable a specific value

examples:

```
assign v6 ~v1
```
or
```
assign v2 300
```
or
```
assign v0 #cur_fuel
```

You may also assign a variable a text value for use in your code for example:

```
assign vh turn right
if scan found enemy then ~vh
```

math v# = (value) (+, -, *, /) (value) **…**
The math statement is used to do math calculations to a variable.

"+" = add
"-" = subtract
"*" = multiply
"/" = devide

Example:

```
math v4 = ~v4 + ~v3
```
or
```
math v4 = ~v4 + #cur_ammo
```
or
```
math v6 = ~v6 / ~v3 + 7
```

Detailed example:

if the value of v6 is 10 and the value of #cur_ammo is 100 and the command read:

```
math v0 = #curr_ammo / ~v6 + 4
```

this would make v0 have a value of 14.

This is what the compiler would see:

```
v0 = (100 / 10) + 4
v0 = 14.
```

Looping Example:

```
gosub loopit


loopit:
assign v1 1
next:
move forward
math v1 = ~v1 + 1
if value ~v1 = 10 then return
goto next
```

High level if statements:

if value (variable) **(**>, <, =, <>, >=, <=**) (**value or variable**) then** [additional command**]**

If the condition of the statement is true then do the command to the right of the word then.

Examples:

```
if value ~v6 > 100 then fire weapon
```
or
```
if value #set_ammo > ~v6 then
      self destruct
end if
```
or
```
assign v3 #scan
if value ~v3 <> barrier then
      raise shield
      move forward
end if
```

Note:high level if statements must start with these two words: **"if value"**

A common mistake in using high level if statements is made when the programmer forgets to use the **value** keyword! Many low level if statements do not use the value keyword and this causes some confusion.

The registered debug watch feature will catch this.

Examples:

Correct Use:

```
if value #set_ammo > ~v6 then
      launch missile
end if
```

Incorrect Use:

```
if #set_ammo > ~v6 then
      launch missile
end if
```

Appendix: Battle Notes

- If you take a flag with full power (no damage) your system will overload and you will take 50% damage.
- If you are damaged and you take a flag all damage will be repaired and 10 ammo will be added.
- If a unit hits a mine it will do 50% of the maximum damage setting to the unit.
- Fuel and power are separate. Fuel is only needed for mobility and cooling for the onboard computer. Running out of fuel only means that your A.I. unit will not be able to move forward, backward or turn.

Appendix: Scanning

The A.I. command language explains the ranges but here is an example:

A long range scan scans forward from the bugs location until it sees an object up to the full length of the battlefield.

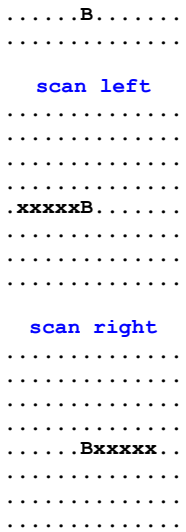
A gps scan scans any designated x and y coordinate given in the command.

Scan examples:

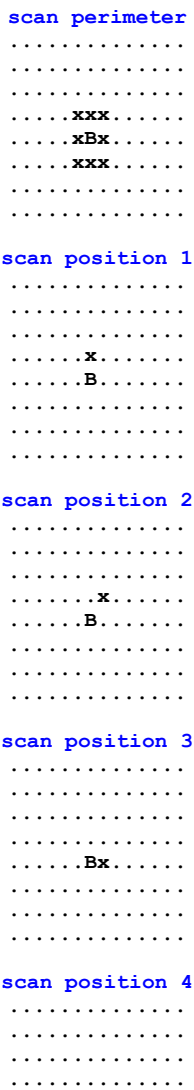
B = A.I. Unit (Cybug)
x = scanned area
. = blank space

Note: In the following examples the Cybug is facing north:

```
scan forward
.............
......x......
......x......
......x......
......x......
......x......
```



The following examples are not heading specific.



```
......B.......
.......x......
..............
..............
```

scan position 5

```
..............
..............
..............
..............
......B.......
......x.......
..............
..............
```

scan position 6

```
..............
..............
..............
..............
......B.......
.....x........
..............
..............
```

scan position 7

```
..............
..............
..............
..............
.....xB.......
..............
..............
..............
```

scan position 8

```
..............
..............
..............
.....x........
......B.......
..............
..............
..............
```

cross scan

```
..............
..............
..............
......x.......
.....xBx......
......x.......
..............
..............
```

corner scan

```
..............
..............
..............
.....x.x......
......B.......
.....x.x......
..............
..............
```

Appendix: Weapons and Damage Summary

Primary Weapon (Projectile Gun)

Range 5 Spaces.
Maximum Damage to unshielded units within 1 space is 5
Maximum Damage to unshielded units within 2 spaces is 4
Maximum Damage to unshielded units within 3 spaces is 3
Maximum Damage to unshielded units within 4 spaces is 2
Maximum Damage to unshielded units within 5 spaces is 1
Maximum Damage to shielded units within 1 space is 1
Maximum Damage to shielded units within 2 spaces is 1
Ammo used when fired is 1
Shields must be down to fire

Missiles

Range Unlimited.
Maximum Damage to Unshielded units is 90%
Maximum Damage to shielded units is 70%
Splash damage to Unshielded units is 60%
Splash damage to shielded units is 40%
Ammo used when fired is 10
Shields must be down to fire

Land Mines

Range is limited to its occupying space.
Maximum damage to Unshielded units is 50%
Maximum damage to shielded units is 50%
Ammo used to produce is 2.

Energy Discharge

Range all spaces surrounding Cybug.
Damage caused to discharging Cybug is 1.
Maximum damage to Unshielded units is 3.
Maximum damage to shielded units is 3.
No Ammo is used to discharge energy.
Energy discharges also destroy Mines and Flags.

Self Destruct

Range all spaces surrounding Cybug.
Damaged caused to destructing bug as well as all surrounding Bugs is the difference between the destructing bugs current damage and the maximum battle damage setting.
Blast Damage to all surrounding A.I.Bots is the same number.
No ammo is used and no flag is left.

System Overload

System Overloads occur when a fully energized unit attempts to add more energy (taking a flag). This causes an overload and will do 50% damage.

Fuel Burn Rate

This is the rate at which a unit burns fuel. The burn rate increases every time a unit sustains damage and or fails at a repair attempt. The burn rate cannot be repaired or slowed during battle.

Damage, Points & Burn Rate Grid

WT = Weapon Type
- PR1 = Primary Weapon Range 1
- PR2 = Primary Weapon Range 2
- PR3 = Primary Weapon Range 3
- PR4 = Primary Weapon Range 4
- PR5 = Primary Weapon Range 5
- MSL = Missile
- MSLS = Missile Splash Damage
- MIN = Mine
- EDC = Energy Discharge
- SDT = Self Destruct
- SOL = System Overload

AR = Ammo Required
DCE = Damage caused to enemy with Shields Up/Down
DCS = Damage Caused to Self with shields Up/Down
EBR = Enemy Burn Rate Increase with shields Up/Down
PTS = Points added to Score of firing unit for hitting enemy with shields Up/Down

WT	AR	DCE	DCS	EBR	PTS
PG1	1	2/5	0/0	2/5	40/100
PG2	1	1/4	0/0	1/4	20/80
PG3	1	0/3	0/0	0/3	0/60
PG4	1	0/2	0/0	0/2	0/40
PG5	1	0/1	0/0	0/1	0/20
MSL	10	70/90%	0/0**	6/9	300/500
MSLS	10	40/60%	40/60%	4/7	200/400
MIN	2	50/50%	0/0	5/5	0/0
EDC	0	3/3	1/1	3/3	100/100
SDT	0	*	*	10	100/100
SOL	0	0/0	50/50%	5/5	0/0

*Variable depending of remaining Energy Points
** Missile will score a direct hit on the firing unit itself, if fired while shields are still up.
You do not get splash damage points for harming yourself.

Other Fuel Burn Factors:

- All forward, backward and turning movements require 1 unit of fuel.
- One Fuel unit is burned every 10 Clicks for the onboard computer cooling device.
- Missiles use 300 fuel to load and Launch.

Other Scoring Factors:

- 50 Points for every life point remaining until maximum death damage and -50 for every damage point over maximum damage.
- Final score equals total score minus the current units burn rate.