Get the Software!

A.I. Wars
The Insect Mind

A.I. Wars
Armor Commander

The most fun you'll have programming!

Available at
TacticalNeuronics.com

A.I. Wars User's Handbook

A.I. Wars

User's Handbook

Published 2005
Printed by CafePress.com in the United States of America

I would like to thank all of the players of A.I. Wars for their input. This game continues to grow and get better with your help.

Happy Hunting!

Acknowledgements

A.I. Wars concept, design and programming, 2D MIDI music and some splash screens textures and artwork:
John A. Reder

Some rendered splash screens and most in game 2D Cybug, effect and environmental 2D graphics and 3D Cybug Flag:
Alvin Helms

CAICL editor design input provided by:
Vidar Braut Haarr

3D Engine MIDI Music by:
PIX

Tournament level Cybugs have been donated by:
Devon Ellis
Tony Dwyer
Marty Lawson
Ted Mitchell
James Press
Shea Parkes
Steven Mast
Esa-Pekka Pälvimäki

Please send any comments, bug reports and suggestions to this E-mail address:
John.Reder@TacticalNeuronics.com

To find out the latest news, tournament sites, Cybug development tips, A.I. Wars forum and A.I. Wars updates visit the official A.I. Wars web site at **TacticalNeuronics.com**

Table of Contents

Section I

13 Our Future
17 What is A.I. Wars
19 How to play: Creating an A.I. unit
20 To edit a Cybug
22 Editor features
23 Verifying your code
24 A.I. file security
25 Preparing for battle
27 Sound effects
28 Music Options
29 During the battle
30 Clicks
31 Overheat
32 End of battle
33 Debugging
35 After the battle
36 Scoring
37 Recorded battles
38 Scenarios
39 Damage settings
40 Map editor
41 Tournament mode
42 Advanced tournament mode
43 Play by e-mail or file transfer

Section II

47 CAICL Quick reference
51 Command language syntax
52 Unit identification and security
54 Debugging commands
55 Program branching
56 Unit rear data resource ports
57 Unit damage status and repairs
59 Randomizing A.I.
60 Scanning for objects and other units
64 Movement and location
67 Weapons Control
71 Protection
73 Nesting
75 User variables
76 System variables
79 Advanced Commands

81 High level if statements
83 Appendix: Battle notes
84 Appendix: Scanning
85 Appendix: Weapons and damage
87 Damage, points and burn rate grid
88 Other fuel burn factors
89 Other scoring factors
90 Advanced cMath finctions

Our Future...

The risks of warfare will be bloodless ones! Our willingness to commit troops will be a simple matter of economy and production. Militaries worldwide are developing automated intelligent fighting machines. These machines will be much harder to stop than the typical human soldier and will be able to react and move with speeds and agility that will be impossible for its human opponents to match or mimic. For example, we'll see aircraft that can travel and turn so fast they'll generate G-forces that would liquefy its pilot (if it had one.) These machines are not science fiction, and are currently in development today. They have no fear, and will show no mercy...

Tactical Neuronics has introduced its new line of automated all terrain six legged fighting machines code named: **Cybugs**. You are now the proud owner of A.I. Wars (The Insect Mind) (Cybug battle simulation software). This simulator allows potential **Artificial Intelligence** (A.I.) programmers to perfect the strategies necessary to take our Cybug A.I. to a level that could adaptively compete on the modern battlefield.

The Cybug is a programmable mobile platform, with six legs that have been incorporated for stability on multiple terrain types, the original two and four legged designs proved too unstable to withstand the blast forces from explosions on or near its center of balance. The Cybug is an all weather assault platform that is loaded with an internally concealed missile and grenade launching system and an external projectile weapon located in the center of its fluid cooled Cybrain appendage (head). The Cybug caries an internal power source that uses a crystalline fuel converter to power its engines and internal electronics. This power system also runs a cooling system that can help keep the Cybug cooled but it can be overheated if items like the shield generator run for an extended period of time. This power system can be discharged through the shielded armor shell of the Cybug, but this discharge will also slightly damage this shell as well (note: This discharge can be set high enough to self destruct the Cybug if the need arises). Its CPU runs single commands transmitted from a remote **Battle Simulation Coordination Processor** (BSCP). The BSCP negotiates commands in packets that are weighted based on their resource consumption. Movement commands are considered heavier than scan commands and logic commands are simple and can be utilized in larger quantities. The internal power source can convert Ammilian alloy (an explosive metallic

substance) into usable ammunition. The Ammilian converter can produce simple projectiles easily but requires more alloy to create items like mines, grenades and missiles. Grenades and missiles also utilize fuel to propel them to their targets. The Cybugs can gather resources (fuel and ammo) while on the battlefield by collecting flagged replenishment packs. Cybugs can also transfer resources to each other by connecting their resource ports located in the rear or the Cybug. There is also a data port that has a communications link where commands can be inserted manually into the Cybugs CPU for maintenance reasons (Warning: our engineers fear that this port may be a security risk if enemy Cybugs can gain access it.)

Cybug squads (Hive formations) are possible since each unit can transmit an IFF code. This code identifies a Cybugs secret team name, multiple Cybugs with the same code are on the same team, and can identify each other as friendly units to avoid accidental friendly fire damage. Cybugs that have the same IFF codes can transmit data to each other using the hive variables on the BSCP. All Cybugs have access to the global BSCP systems system variables that are considered common knowledge among all Cybugs in a simulation.

Goal orientation can be utilized in a simulation via the use of Queen Cybugs and the Strategy Node. The BSCP can be set to terminate the battle when a Queen is killed, typically awarding the killer with a large number of points. The strategy node can be used as a refueling and ammo depot as well as its ability to grant award points to any Cybug occupying it for a predetermined amount of time.

Your goal as a Cybug A.I. developer is to develop competent Cybug reflexes, plus offensive and defensive strategies to ensure the survival of your Cybug or your team of Cybugs in a battle simulation. You will be able to develop their A.I. using the Cybug A.I. Command Language (CAICL) provided. The A.I. editor comes complete with a description of each command, as well as code verification and debugging tools. Cybug A.I. files can be encrypted and password protected for transport outside of the BSCP.

What is A.I. Wars?

A.I. Wars allows you to design the **Artificial Intelligence** of an insect-like mechanized warrior (Cybug) and send it into battle simulations to test its wits against others.

This game is about programming. There I've said it, yes, programming. Now don't get discouraged; this is a game, not a computer science course. The Cybug A.I. Command Language (CAICL) is easy to learn, and simple to apply. Some of the most fearsome Cybugs can be created with a just few lines of well thought out code.

The goal of playing A.I. Wars is to have fun. This is not an action game that involves hand/eye coordination, but rather a game of logic and strategy that is designed to give you the pleasure every programmer gets when they write a successful and efficient program and the excitement of watching your creation pound its competition.

A.I. Wars will give you the basic tools to design your Cybug and experiment with it in varied battle situations. This game has multiple options to define the layout of each battle simulation. It is designed to encourage friendly competition between A.I. designers with options like encrypted ASCII A.I. files for file trading that will allow you to hold tournaments and contests.

Remember that "Smart bugs never die" and "May your A.I. Bugs be without A.I. Bugs!"

How to Play

Creating an A.I. unit

Example Cybugs are included with A.I. Wars. These Cybugs are basic and are designed to let you see how Cybugs may be programmed. A few tournament level Cybugs have also been donated to give you an idea of just how competitive some Cybug developers are. The scenarios that are included will utilize these example Cybugs and give you basic opponents that can be pitted against your home-grown Cybugs. Once your home-grown Cybugs are developed to the point where the example Cybugs present little challenge, then you are ready to send them to your friends via E-mail, BBS's, FTP and Web sites to participate in contests and tournaments.

To Edit a Cybug

1. Click on the Program A.I. button in the Battle setup screen or the Edit button in the main title screen.
2. Select File, then New or Open to start editing your Cybug.
3. Develop the Units A.I. using the commands and syntax of the A.I. command language. (Save your work periodically using the save file button.) Note: there is a 2000 line limit on A.I. file length.

Note: You may also launch A.I. Wars with the /E: option to load the editor with the specified A.I. file upon start. Example: AI_WARS.EXE /E:droneA1.ai

Example A.I. Code: Notice how simple the code is. One mistake many Cybug developers make is to get lost in sophistication and forget about survival and offense. A good basic strategy and compact code are usually the best ingredients for success. As you can tell by its name, his primary job is to defend and evade.

```
name survivor
iff code x111
author Tactical Neuronics

raise shield
phase1:
long range scan
if scan found enemy then
        gosub killit
end if
scan perimeter
if scan found enemy then discharge energy
if bump barrier then move backward
turn right
if damage is > 1 then attempt repairs
goto phase1


killit:
lower shield
if missile ready then launch missile
raise shield
turn left
move forward
return
```

4. Once you are done programming select the Save button.

Note: to edit the file after you have left the edit screen you may choose which file to edit by clicking on its name in the

Battle setup Cybug file list and select the Program A.I. button. This will load the selected A.I. file.

Editor Features

The editor has some built in features to make it easier for you to use the CAICL. The main feature is the CAICL Help button, this will give you a quick reference to the language and its syntax. The CAICL Help window has a drop down text box titled 'Table of Contents' which will jump to the section of the CAICL reference that you need. You can also search for a specific string by typing it into the text area given and clicking on the find button.

Another feature in the editor is the syntax help line given at the top of the code window. This line allows you to type in the first few characters of a code string and will attempt to finish it for you as well as provide a guide to its syntax. You can click on the insert button or press enter to insert this code into your code text at the current location of your code text cursor. You can search for any text string that you type into this box by clicking on the find button. Note: to highlight all of the text in this box double click on it. To do cut and paste anywhere in the editor simply right click in the text windows provided.

Verifying your code

Your code can be compared to a syntax checking engine that will turn your correct text to blue and your variables red and comments green when you hit the verify button at the top of the editor window. This should help you catch simple keyword and variable misspelling errors in your code. For advanced users, you may choose to use the CAICLPro editing software available from **TacticalNeuronics.com.**

A.I. File Security

If an A.I. file uses the password command the file will not allow you to edit it without entering the Security Password in the Security Password dialog window.

Preparing for Battle

A battle requires the following options:

- One or more A.I. units in the **Cybugs in Next Battle** List.
- A Map Selected in the **Battle Map** field.
- Battle Options load to the default or last battle settings. You can change these settings by selecting the desired option check boxes as well as clicking on the **Music, video and other settings** button. Here are some of the more common options:
- The **Starting Ammo** option allows you to set the basic ammo load of the Cybugs when the battle starts. Cybugs can gain more ammo after the game starts when they gather flags.
- The **Maximum Damage** field. (a Cybug dies when its damage reaches the maximum damage setting. 99 is the maximum setting.)
- The **Starting Fuel** Field. (A Cybug burns fuel for movement. Damage to a Cybug causes the Fuel to burn faster as the fuel cells and motors are damaged. These cannot be repaired during battle.) Note: a small amount of fuel burns every so often due to the Cybugs need to cool his onboard computer.
- Note: These options can be set automatically for you if you select a scenario from the Scenario list. You can save your battle settings by choosing the **Create Scenario** button. This creates a scenario (.SCN) file. (see scenarios.) Other options include **Disable IFF Codes, Disable Strategy Node, Position Strategy Node, Blood Mode, One click one command, Disable shield overheat, End Battle on Queens death...** May be also be selected.
- The **End battle if Cybugs stop roaming** option will force the battle to end if the Cybugs stop moving forward or backwards for 100 clicks. This prevents the battle from degrading to a point where the only Cybug(s) with fuel are camping in one spot. When this occurs the battle can last a very long time since the only fuel drain is on every 10th click.
- **Advanced Battle settings** can also be set allowing you to dictate the types of damage and fuel and ammo costs for each weapon and scoring values.
- You have four possible battle view options: **2D Birds Eye, 2D Follow, 3D and Text Only**. The 2D Birds Eye view shows you the complete map (unless the map resolution size exceeds your current screen size. The 2D Follow Cam shows you a transmission from your Cybug, which displays his surroundings. This view stays centered on the selected

Cybug and follows it as it roams around the battlefield. The 2D modes are best used for Cybug development. Text mode simply shows you the battle summary text as it is being created (Note: This is the fastest mode). 3D mode will script the battle and launch the 3D Battle Viewer. This mode is the most realistic. All battles can save their results to a 3D script for viewing at a later time with the recording viewer if desired.

- Once all Fields have the desired values select the **Start Battle Simulation** button to start the battle.

Sound Effects

If your PC has the proper hardware and drivers for sound you have the option of having digitized sound effects.

Note: You may use the Text Mode (No battle graphics) check box to speed up the battle sequence. This will replace the battle mode display with a text play by play depiction of the battle as it happens.

Music Options

You are given 3 Music options from the battle setup screen.

1) MIDI (this plays MIDI files located in the A.I. Wars home directory.)
2) CD Audio (This plays any CD you have in your CD ROM Drive.)
3) No Music – this option may increase the speed on some versions of windows. If A.I. Wars is too slow, try this setting.

Select which ever option you desire. The selected Music will play during the battle simulation mode.

During the Battle

Closing it using the Cancel Battle button in the Birds Eye view can stop the 2D battle. You are also given the chance to toggle sound effects, pause music and show unit names using the buttons displayed above the Birds eye battle view window. If you are using the Follow cam you can get to the Birds Eye view by clicking on Follow Cam 'close' button.

To cycle to another Cybug in the Follow cam you can click on the appropriate Cybug number button. Dead or unused Cybug buttons will be disabled.

If the **Show Unit Names** option is selected in the Birds Eye view the Cybugs will have their names next to them. They will have a default name of AI# and their assigned number unless the A.I. file contains the name command. Following their name you will see their ammo, damage, remaining fuel and their heat level.

Example: #1 Crusher (15,3,1700,11)

The 3D viewer lists its available keyboard commands in the top left corner. You have the choice between following a specific Cybug (by selecting its number or the FREE CAMERA mode (ctrl) which allows you to roam the battlefield with a free-floating camera. Note the (Page Up) and (Page Down) keys control camera up and down and the arrow keys move forward, left, right and backwards while the mouse allows you to control where the camera points. The (P) key can pause the battle and the (Esc) key will exit the battle at any time.

Clicks

Clicks are the time slices in a battle. All Cybugs share these time slices on the CPU where all Cybugs get to execute at least one command every click. In the one click one command mode each Cybug gets to issue one line of code per click. If the one click one command (OCOC) option is not set then the Cybugs can execute multiple commands in a single click (Smart Clicks). Logic commands take only 1/10 of a click and scan commands take 1/3 of a click and movement, firing weapons, discharging energy, cloaking and attack commands take 1 click.

Hint: one command line can contain many commands as long as they fit into the syntax of a single command string for example:

if value ~v1 = 10 then if not bump barrier then if ammo is > 10 then launch missile.

Overheat

Overheating can occur if your Cybug uses its shields for a time period longer than the maximum overheat click setting. For example if a Cybug uses its shield for 30 clicks without turning it off and the overheat setting is 30 then the Cybug will go offline and drop its shields for 10 clicks, during this time the Cybug cannot do anything but sit there and hope nobody attacks it. The Cybug gains 1 heat for each click its shields are raised and looses one heat (cools) for every click its shields are lowered.

End of the battle

The battle ends when one of these conditions are met.

- All A.I. units run out of fuel
- When only one team remains
- The user cancels the battle.
- A Queen Cybug has been killed and the **Battle ends on Queens death** option is set.

Debugging

You will be able to debug your Cybug by clicking on the Cybugs file name in the battle list before starting the battle (This will place a red flag on the selected unit so you can follow it as it moves through the battlefield. (The Follow Cam will default to this Cybug when the battle begins.) When the unit fires or launches its weapons, enables shields or scans you will see the actions represented in the battles graphics. If you selected the **Debug Mode on for marked Cybug** check box the debug data will be displayed in battle including the current line of code being processed by the selected unit. When debug is on, the battle can be slowed down using the scrollbar at the top of the display so you can have time to analyze the debug information. A special map has been created called **debug.map** to help you see the data better. This map will not allow units and graphics to enter the debug display area. Note: You may use any map but the debug map is recommended for easier viewing.

You may place the commands **debug on** and **debug off** in your Cybugs code. When activated, this will save any code that the unit sees into an area called the **Debug Watch buffer**. The buffer will display any commands that the unit sees and its reactions to the commands including error and warning messages about possible A.I. Command language errors. You can view the buffer contents by choosing the **debug watch** window button in the battle summary screen.

Note: The Cybug must be marked by clicking on the Cybugs name in the battle list found in the battle setup screen (a yellow light will be on next to the selected Cybug).

Example Debug Watch Buffer Information:

```
CLICK  SC LINE COMMAND
0      4  8    start:
0      5  9    raise shield
1      0  10   generate random
1      1  11   assign v0 0
1      2  12   goto hunt
1      3  14   hunt:
1      4  15   math v8 = 18 - 22
1      4  15   assign v8 -4
1      5  16   math v9 = 11 - 8
1      5  16   assign v9 3
1      6  17   if value -4 > 0 then goto gowest
1      7  18   if value -4 < 0 then goto goeast
1      7  18   >>> condition met executing command:
goto goeast
1      7  18   goto goeast
1      8  76   goeast:
1      9  77   if facing east then goto e1
1      9  77   >>> condition met executing command:
goto e1
1      9  77   goto e1
2      0  82   e1:
2      1  83   long range scan
2      3  83   >>> scan identified: mine
2      4  84   if scan found enemy then goto missle
2      5  85   move forward
3      0  86   if bump barrier then math v0 = 0 + 1
3      1  87   if value 0 > 6 then goto ramble
3      2  88   goto hunt_y
3      3  19   hunt_y:
3      4  20   if value 3 > 0 then goto gonorth
3      4  20   >>> condition met executing command:
goto gonorth
3      4  20   goto gonorth
```

After the Battle

The Battle will display the battle statistics. They give a description of the battle settings, a play by play description of the battle, and the Cybug individual/team standings and scores.

You can **Print** this or **Save** this to a file from this screen.

You can view this screen later using the **Last summary** button from the main battle setup screen.

Scoring

Units get points for inflicting damage on other units while keeping their damage low. To gain extra points program your Cybug to locate the Strategy Node and have it go there. Cybugs sitting in the strategy node can gain points, ammo and fuel for each click while sitting in the node.

Team scoring consists of the entire point collection of each Cybug containing the same IFF code. Cybugs with the same IFF code are considered members of the same team!

Recorded Battles

The battle recorder allows you record and playback battles. The recorder can be found by clicking on the **Settings and recorder button** in the battle setup screen. You have 5 controls to choose from that pertain to the recorder, they are:

1) The check box that tells A.I. Wars to **save 2D battle recording to a file**.
2) The field that allows you to input the name of the 2D recording (.rec) file.
3) The check box that tells A.I. Wars to **save 3D battle recording to a file**.
4) The field that allows you to input the name of the 3D recording (.3dr) file.
5) A button that opens the battle replay utility (**View Recorded Battle**).

The battle replay utility allows you to select a recording (.rec or .3dr) file for replay and run it. The 2D battle replay looks a little different than the actual battle. This utility has 2 windows:

1) The battle select/start/mark Cybug and statistics window.
2) The battle replay window.

Note that during the 2D battle replay an energy discharge is a red flash and a fire weapon is a white flash on the Cybug. The selected Cybug has a green circle over it.

Scenarios

Scenarios allow you to select from a list of pre-selected battle configurations. These configurations include a group of Cybugs, a battle map and preset ammo, damage and fuel settings. A.I. Wars comes with some default scenarios to give you some basic battle settings in which to test your Cybugs.

You may save any battle configuration by choosing the **Create Scenario** button.

Damage Settings

You can change the damage caused by actions like landing on a Mine, Overloading your Cybugs energy supply, Primary weapon hits and Missile Hits by selecting the **Settings and Recorder** and **Advanced settings** buttons.

Note: The Advanced Settings apply to all scenarios and battles. You can return to the default settings by selecting the **Restore default values** button in the advanced settings window.

The Map Editor

You can Edit Maps to be used by the Battle Simulator by selecting the **Map editor** button.

The editor has a pull down window that allows you to select the graphics that you want to add to the map. Use the mouse left click to place the graphics and right click to remove them.

Note: the unit start graphic depicts where the units will start when the map is used in battle. Only the first 10 will be used. If starting positions are not set in the editor then the battle simulator will use default starting locations. If you right click on the Cybug start graphic box you will be given the option to actually set the number of the Cybug that will start in a specific location. This allows you to setup cool team battle maps and define Queen Cybug locations etc.

From the pull down window you can **save**, **create a new map**, **clear the map** or **exit the editor**.

Tournament Mode

This mode allows you to play any of the Cybug files in the game directory (up to 500 A.I. files) against each other. The final results will be displayed in the battle statistics screen at the end. Also the entire statistics and play by play readouts will be saved to a file named **contest.txt** *(you have the option to rename this file in the tournament screen)***.** You could post on the web, mail or E-mail this file to the authors of the competing Cybugs.

Note: You are given the option to disable IFF Codes during tournaments. This disables the Cybugs ability to recognize friendly units.

Advanced Tournament Mode

This feature is added to allow tournament sites the freedom to come up with their own tournament engine. The A.I. Wars executable AI_WARS.EXE has three possible command line parameters that will allow batch automation. These parameters are: /S, /B and /M.
/S: = scenario file name (.scn)
/B: = battle summary text file name (.txt)
/M: = movie recording file name (.rec)

Example:
AI_WARS.EXE /S:battle08.scn /B:battle08.txt /M:battle08.rec

An advanced tournament batch creator could create scenarios and then launch the AI Wars engine to run the battle and then look at the battle summary to create the next scenario and continue this loop until the tournament is done.

Play By E-Mail , File Copy or Transfer

The tournament mode is a great way to hold contests for all of your friends' Cybugs. You can have them give you their Cybug files on disk or have them Upload or E-mail them to you. The A.I. files are encoded but they are pure ASCII text so you can just E-mail the text and cut and paste it into a text file with the .AI extension or import the file using the File / **Import encrypted file** option in the Cybug A.I. file builder window.

You may use the File / **View Encoded Version** button in the A.I. editor develop mode screen to view the A.I. file in its encoded form. You may cut and paste this into your E-mail or text editor for distribution if you wish.

Example ASCII encoded AI file:

You can place any comments you like here before the <v4ai> tag . This text is striped from the file if it is edited.

```
This is the encoded ASCII version of the specified
A.I. File.
You may cut and paste this data into E-Mail messages
etc. for A.I. File distribution.

<v4ai>
b½RcñrXVÖe)óXeñbzùcqjYf½e£cXy!b!YeMecMmUzÜcqKYñÑez«eú,
Z½dc.mYpñf£]h/$S½RcWjbL%WWKZ!ZYpÿe?¼Y¢$S£pR£¼V?@c{_ZpP
aLnY£&X)óZ)íZprV?!c{*a£aVV°Zz¿Qy°WyÑXV^dpæ
dÑqW{lSfcT.ZW£ÑgLóXf^cqhbMgXe°Wz°gLnc{nc.lcqra¥dc?YOfÑ
e{Uf)pb.mUWaYpÒgz}Zpsc)æc{&c_cbqZVÑOYpƒf?Uefyb.fbWjY£ñ
fyÑXVnb¥Xc_obWga¥SYzYOf¥ez«fK,cWncqrW£¿fyÑXVvb£nd¥Za.i
U!Rd½cYfáX{ófLrcÑxUWoX?½gzLgz@dqSV.bb¥ZcWkT£VY{»efÖXf%
c!fbqoO(ñf£{Zp^b?ædz&bWjc¥Zc?rXz¿efÑXf&cqdb£%X{¼Y?}fñz
b!Kc)&XV%WLvegPXpÿX{æf{%c£(cqdX)«hKÑXVvb£nd¥Za.iU!Rd½c
YfáX{ófLrcÑxUWoX?½gzLh{*dWRV.ob!bb.fT£VX{óX{ƒd£nc£(
bWjY£ñfzLgf!bÑQe)&c_cbqZVÑOYpƒf?UeV!cMldLlOfùfp|hU,c)ò
c.qSfib½RVÑaXV¡f)Uef@d.ma£%XzófV½Zp&cVòdz&b!aU!PcqUW£ú
e?Uae,W{(c_cXpñY?¿hñ%c{ndgbbWoS)ZdWNQz¥ez«fK,cWnc.iXfÑ
f)ógfzdñneMebMiU!feqZYfTfpóef&Tfmb!aQzùfp|hU,b¥SeWka£%
bqZc½UZpTgVÑeV!VWfcMoYpNf)ógfzdñUT!fbV%bgMd)òX)íX{ªf(,
YV(Vñ%Yñ«f)óZpndqXc_jcWoU!dc½XW£Ñf£ƒVo!cÑnc½ZQz¼g)ªiLn
dVöT{kb!aU!NeqUYzTep«fz%c!dcp%Yñ«f)óZptcÑXd)&a½Vc.ddWM
Y/¢V)ñfV&dL(c!oW£¿hKÑXU$S{ÆcWocqdbqdYzYOfúe?ÖeV%b.sbL%
Y?]gz}hfzTLRc¥cUWma¥Zc.WYVTfLƒXeñVWsbqZYfNf{úh{@U¥VcWk
a¥jcV+T£VX{óX{¥d{!b¥nbñ%X{ùY?dZp&cVòdz&bgjcÑaVÑZW£ÿe)Ü
e¢ÑTfmb!aQz¿fVógV@c?nc¥pUV<U!fdMMYfTeñÜf?@VWqa.iXfÑgpe
Xy!cfAV.oa.ibgad)rX{¥X{oXf&cqdb£%X)ÑhLúZp%b!Rc!lbñ|S)Z
cqNYñíf£¥d{!b¥nbñ½O(ñf£{Zpod!Qd?&a½Vc.ddWMY?TgVÑeV!VWs
c.mYfNgf[gp&TLRc½lc_jU!eefòY?ƒVúÖVo!cMmbMhZpnW[ógf@d?ò
dñ&c!cb½Qd!LO(ÿe{ªfzrVWvbMVYzÑgyÑXV%b!Md¥bUWnb.Uc½TXe¢
V)ƒd£nc£(bWjY?¢fVªgU$S½Mc?&c!Xa¥ZVÑNYpæf)íXfrc¥dbÑtQz¡
g)¿he,bÑRc_jdLlSgSd¼bYpTgLæd{%d)pR£lOf¿fVógV@c?%YfnR¥o
c¥dd?rYLíe{æVo!cÑnc½ZQz¼g)ªiLndVöT{kbgjcÑaVñòXzƒe?¥fzn
c¥ccMhO(ñW[óh)nc{öd_jV£½S)ZegcY?ÿX{¥e)tcqsSfiYVÑhf[Zps
cÑVeqXcqYS)Zd_WZLíX{úfV%dÑZcqYO(ñf{úh{@U£æc.qbMmc.Md.L
YpùVúÖVo!dqZb¥YYpúa)lXy!dqYdÑkUWgbqRefYOfùf?ôeU,cWncqr
W£¿fyÑXVzcÑZc)&bWjc.icqZXe¢V)ÿfV(cL(bWjY?¢fVªgU$S½Kd_q
cL%a¥RegMY?£efÖeL@cñpR£lOf¿fVógV@c?*YfnR¥oc¥dd?rY?ÑeñÑ
f/$S¥lcMqXpNf?úh))b!VcznR¥hcqhc{r
```

Check out the A.I. Wars web site for additional A.I. Wars files and upgrades that you can use in this game. We encourage players to develop swap areas and sites to distribute Cybug .AI files freely.

Note: the map files and scenario files are also in ASCII text formats so you can transfer them the same way as the .AI files.

SECTION II

Cybug A.I. Command Language (CAICL)

1.1 Quick Reference

General Commands:
; [comments]
assign v# ~v#
assign v# [any value]
assign v# #[system_variable]
attempt repairs
author [developer's name]
beep
cloak on
cloak off
cmath v# = (simple or complex math formula)
cross scan
corner scan
data link command [additional command to insert]
data link resource fuel (value)
data link resource ammo (value)
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 not 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 not facing north then [additional command]

if not facing south then [additional command]
if not facing east then [additional command]
if not facing west then [additional command]
if fuel is > # then [additional command]
if fuel is < # then [additional command]
if fuel is = # then [additional command]
if grenade ready then [additional command]
if grenade not ready then [additional command]
if missile ready then [additional command]
if missile not 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
launch grenade
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]
queen
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
scan relative 1
scan relative 2
scan relative 3
scan relative 4
scan relative 5
scan relative 6
scan relative 7
scan relative 8
self destruct
set grenade fuse (value or variable)
turn right
turn left

User 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

Hive (Team) Variables:
h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, ha, hb, hc, hd, he, hf, hg, hh, hi, hj, hk, hl, hm ,hn ,ho ,hp, hq, hr, hs, ht, hu, hv, hw, hx ,hy, hz

System Variables:
#alive
#bloodmodeon
#burn
#cloakstatus
#cur_ammo
#cur_fuel
#cur_head
#cur_life
#cur_score
#damset1
#damset2

#damset3
#damset4
#damset5
#damset6
#damset7
#damset8
#damset9
#damset10
#damset11
#damset12
#damset13
#damset14
#damset15
#damset16
#damset17
#damset18
#enemy_d
#enemy_h
#enemy_x
#enemy_y
#friend_d
#friend_h
#friend_x
#friend_y
#heatlevel
#inbound
#lastcybugon
#max_fuel
#max_life
#overheatoff
#overheatset
#random
#scan
#set_ammo
#shield
#smartclickoff
#snodeoff
#strat_x
#strat_y
#x_pos
#y_pos

2.1 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.

2.2 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:

author 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.

Units with matching IFF Codes are considered to be on the same team. The team score is the total score of all Cybugs with matching IFF codes.
Example:

iff code skull crushers

queen this mutates a perfectly innocent Cybug into a Hive Queen! Queens are designed to be protected because the battle can be set to end upon her death. The dead queen penalizes her team with a point deduction to her score. Queens can be used as battle goals like a flag would be in a capture the flag game.

Team games can be set up with Cybugs and Queens on each team using the same iff codes and team variables to transmit the Queens location to assist in her protection. Example:

queen

2.3 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 A.I. interpreter.
Example:

; this is a comment line

2.4 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

2.5 Unit Rear Data and Resource Ports

data link command [command]
This inserts the command into the opposing Cybugs program at the current program read marker, it can be used against friendly and enemy Cybugs. This will replace whatever command existed in that memory space with the new one which could damage the opposing Cybugs overall program.
Note: this link is only possible if the two Cybugs are touching rear to rear.
Examples:

Data link command iff code myteamiff

Data link command if missile ready then self destruct

data link resource fuel (value)
data link resource ammo (value)
This removes specified resources from the opposing Cybug, it can be used against friendly and enemy Cybugs.

Note: this link is only possible if the two Cybugs are touching rear to rear and shields are down on the initiating Cybug. If the specified amount is larger than the Cybugs current reserves then the remaining reserves will be depleted and the linking Cybug will get what is available.

Examples:

Data link resource fuel 300

Data link resource ammo 25

2.6 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 sign "%". 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 sign "%". Valid percent entries are 0 through 99
Example:

if fuel is < 99 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, a repair attempt will be ignored if the shield is raised.

A successful repair attempt will lower your damage by one point.

WARNING: There is also a 5% 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.

2.7 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

2.8 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.

scan relative 1 - This scans one space directly in front of the unit.

scan relative 2 - This scans one space directly front right of the unit.

scan relative 3 -This scans one space directly right of the unit.

scan relative 4 -This scans one space directly rear right of the unit.

scan relative 5 -This scans one space directly behind the unit.

scan relative 6 -This scans one space directly rear left of the unit.

scan relative 7 -This scans one space directly left of the unit

scan relative 8 -This scans one space directly front left 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 then 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 DroneD2.ai program is an example of how to utilize these variables.

Example:

if x coordinate is = #enemy_x then turn right

2.9 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 not facing north then [additional command]
if not facing south then [additional command]
if not facing east then [additional command]

if not facing west then [additional command]
If the unit is not facing north, south, east or west then execute the command to the right of the word then.
Example:

if not 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

if not bump barrier then [additional command] - Use this to check and see if movement is not blocked by a barrier. If a barrier is not blocking the units path then it will execute the command to the right of the word then.
Example:

if not bump barrier then turn right

Advanced Command:

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

2.10 Weapons Control

Weapons use ammo and possibly fuel to use. The default settings are shown in an Appendix below. These settings can be changed.

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 in front of the Cybug. 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 an 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 or the missile will misfire doing 90% damage to the launching unit.

Note: Missiles require 300 fuel and 10 ammo to fire. These damage and requirement settings can be changed.
Example:

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

launch grenade - Launches grenade. The grenade will travel in the direction fired until a object is hit or its fuse setting has been reached. grenades do 70% damage to units that are hit with their shields down. They do 50%

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 grenade with your shields up. The grenade will misfire doing 70% damage to the launching unit.

Note: Grenades require 200 fuel and 5 ammo to launch. These settings can be changed.

Example:

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

set grenade fuse (value or variable) - This will set the distance a grenade travel before it detonates. Grenades will not travel less than 2 spaces unless they hit another object.
Example:

```
set grenade fuse 12
```

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 two points of damage (note: these damage settings can be changed). An 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 Cybugs 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.

if missile not ready then [additional command]
If the missile doesn't have enough fuel and ammo to launch then do the command to the right of the word then.
Example:

if missile not ready then fire weapon

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

if grenade ready then launch grenade

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

if grenade not ready then [additional command]
If the grenade doesn't have enough fuel and ammo to launch then do the command to the right of the word then.
Example:

if grenade not ready then fire weapon

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

2.11 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. Using the shield causes your Cybugs power source to generate heat. Overuse of the shield can cause your Cybug to overheat. If your Cybug overheats your power source will force a shutdown in order to cool off. Your Cybug will be unable to function until it's power source has cooled off and your shields will shut off until another raise shield command is executed.
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.

cloak on
This turns on the Cybugs cloaking device, enabling it to avoid nearest enemy scans using the enemy_x, y, d and h variables.
Note: This device consumes 10 fuel and two ammo per click to run.

Nearest enemy scans will return the nearest enemy that is not cloaked or will return a zero if the only enemy remaining is cloaked.
Example:

Cloak on

cloak off
This turns on the Cybugs cloaking device off. This will happen on its own if the Cybug runs out of enough fuel or ammo to sustain it or if the Cybug uses a weapon.
Example:

Cloak off

3.1 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.

3.2 User Variables

You have 41 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

team variables include:
h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, ha, hb, hc, hd, he, hf, hg, hh, hi, hj, hk, hl, hm ,hn ,ho ,hp, hq, hr, hs, ht, hu, hv, hw, hx ,hy and hz

Note: team variables are shared by all Cybugs with the same iff code.

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:

3.3 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
#strat_x	x position of strategy node
#strat_y	y position of strategy node
#alive	Number of remaining Cybugs
#inbound	Inbound enemy missile or grenade 0 = no inbound object 1 = from the north 2 = from the east 3 = from the south 4 = from the west
#heatlevel	Current Cybug heat level
#cloakstatus	0 = off 1 = cloaked
#overheatset	Current overheat setting
#overheatoff	0= Overheat possible 1=off
#snodeoff	1=Strategy node off 0=on
#smartclickoff	1=1 command 1 click 0=10 logic commands or 3 scan commands In a single click
#bloodmodeon	1=bloodmode enabled 0=off
#lastcybugon	1=last cybug bonus on 0=off
	Battle damage settings
#damset1	Mine
#damset2	Missile direct hit
#damset3	Missile fragment
#damset4	Primary weapon range 1
#damset5	Primary weapon range 2
#damset6	Primary weapon range 3
#damset7	Overload
#damset8	Energy discharge on enemy
#damset9	Energy discharge on self
#damset10	Missile direct hit with shield
#damset11	Missile fragment with shield

#damset12	Primary weapon range 1 shield
#damset13	Primary weapon range 2 shield
#damset14	Primary weapon range 3 shield
#damset15	Primary weapon range 4
#damset16	Primary weapon range 4 shield
#damset17	Primary weapon range 5
#damset18	Primary weapon range 5 shield
#damset19	Grenade direct hit with shield
#damset20	Grenade direct hit
#damset21	Grenade fragment with shield
#damset22	Grenade fragment

Note: #inbound will report an inbound missile or grenade even if its path to your Cybug is blocked.

Note: #strat_x and **#strat_y** will both return a value of zero if the strategy node is turned off in a tournament.

3.4 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. It calculates from left to right.

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

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
```

cmath v# = (simple or complex formula) **…**
The cmath statement is used to do complex math calculations to a variable. It calculates using advanced math rules. It calculates within parentheses and does calculations on division and multiplication first.

Example:

```
cmath v4 = (~v4 + ~v3) * 3
or
cmath v4 = ~v4 + (#cur_ammo / 2)
or
cmath v6 = ((~v6 / ~v3)+(7 / 2)* 3)
```

3.5 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
```

4.1 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 30 ammo will be added along with 350 units of fuel.
- If a Cybug hits a mine it will do a percentage of the maximum damage setting to the unit.
- Fuel and energy 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.
- Strategy nodes will give the Cybug one point for every click that the Cybug occupies the square. If the Cybug can hold this position for most of the battle this can add up to thousands of points!
- Map coordinates (top left corner = x:1, y:1, bottom right corner x:43, y:30)
- In the past all commands took one click each, this put smarter Cybugs at a disadvantage, so now A.I. Wars calculates clicks in the following order: Movement and weapon command lines take up a single click, Scanning takes up about a third of a click and logic commands take up one tenth of a click per line with the exception of blank lines, labels, name, password and author commands which take up no part of a click at all. To make your Cybug more efficient you can stack commands when possible into a single command line, for example: (Note: this example would appear on a single line in the AI code.)

  if facing north then if ammo is > 10 then if fuel is > 99 then fire missile

4.2 Appendix: Scanning

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 are shown in the ‘View Cybug’ option, found in the title screen.

4.3 Appendix: Weapons and Damage

Note: Damage values reflect default settings. These settings can be changed.

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 2
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 60%
Splash damage to Unshielded units is 70%
Splash damage to shielded units is 40%
Ammo used when fired is 10
Shields must be down to fire

Grenades
Range Unlimited. Range set by fuse.
Maximum Damage to Unshielded units is 70%
Maximum Damage to shielded units is 40%
Splash damage to Unshielded units is 50%
Splash damage to shielded units is 20%
Ammo used when fired is 5
Shields must be down to launch

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 2.
Maximum damage to shielded units is 2.
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
PW1 = Primary Weapon Range 1
PW2 = Primary Weapon Range 2
PW3 = Primary Weapon Range 3
PW4 = Primary Weapon Range 4
PW5 = Primary Weapon Range 5
MSL = Missile
MSLS = Missile Splash Damage
GRE = Grenade
GRES = Grenade 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
PW1	1	2/5	0/0	2/5	40/100
PW2	1	1/4	0/0	1/4	20/80
PW3	1	0/3	0/0	0/3	0/60
PW4	1	0/2	0/0	0/2	0/40
PW5	1	0/1	0/0	0/1	0/20
MSL	10	60/90%	90%/0	6/9	300/500
MSLS	10	40/70%	40/70%	4/7	200/400
GRE	5	40/70%	70%/0	12/18	600/1K
GRES	5	20/50%	20/50%	2/4	100/200
MIN	2	50/50%	0/0	5/5	0/0
EDC	0	2/2	1/1	2/2	100/100
SDT	0	*	*	10	100/100
SOL	0	0/0	50/50%	5/5	0/0

*Variable depending of remaining Energy Points
** Missiles and grenades will score a direct hit on the firing unit itself, if fired while shields are still up.

Note: You do not get points for harming yourself with a misfired or closely detonated missile.

Other Fuel Burn Factors:

- All forward, backward and turning movements require fuel at a cost of one burn rate cycle. Cybug movements become more costly as it's burn rate increases.
- One Fuel unit is burned every 10 Clicks for the onboard computer cooling device.
- Missiles use 300 fuel to load and fire.
- Grenades use 200 fuel to load and launch.
- Cloaking uses 10 fuel per click.

Other Scoring Factors:

- 50 Points for every life point (number of remaining points between current Cybug damage and Maximum damage) remaining and -50 for every damage point over maximum damage. This can be turned off using the **Blood Mode** option.
- Final score equals total score minus the current units burn rate.

Advanced cMath Functions

These functions can be embedded within cmath commands.

ABS

Returns a value of the same type that is passed to it specifying the absolute value of a number.

Syntax

Abs(number)

The required number argument can be any valid numeric expression.

Remarks

The absolute value of a number is its unsigned magnitude. For example, ABS(-1) and ABS(1) both return 1.

ATN

Returns the arctangent of a number.

Syntax

Atn(number)

The required number argument is any valid numeric expression.

Remarks

The Atn function takes the ratio of two sides of a right triangle (number) and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The range of the result is -pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Note: Atn is the inverse trigonometric function of Tan, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse Atn with the cotangent, which is the simple inverse of a tangent (1/tangent).

COS

Returns the cosine of an angle.

Syntax

Cos(number)

The required number argument is any valid numeric expression that expresses an angle in radians.

Remarks

The Cos function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

EXP

Returns e (the base of natural logarithms) raised to a power.

Syntax

Exp(number)

The required number argument is any valid numeric expression.

Remarks

If the value of number exceeds 709.782712893, an error occurs. The constant e is approximately 2.718282.

Note: The Exp function complements the action of the Log function and is sometimes referred to as the antilogarithm.

INT, FIX

Returns a value of the type passed to it containing the integer portion of a number.

Syntax

Int(number)

Fix(number)

The required number argument is any valid numeric expression. If number contains Null, Null is returned.

Remarks

Both Int and Fix remove the fractional part of number and return the resulting integer value.

The difference between Int and Fix is that if number is negative, Int returns the first negative integer less than or equal to number, whereas Fix returns the first negative integer greater than or equal to number. For example, Int converts -8.4 to -9, and Fix converts -8.4 to -8.

Fix(number) is equivalent to:

Sgn(number) * Int(Abs(number))

LOG

Returns the natural logarithm of a number.

Syntax

Log(number)

The required number argument is any valid numeric expression greater than zero.

Remarks

The natural logarithm is the logarithm to the base e. The constant e is approximately 2.718282.

You can calculate base-n logarithms for any number x by dividing the natural logarithm of x by the natural logarithm of n as follows:

Logn(x) = Log(x) / Log(n)

RND

Returns a random number.

Syntax

Rnd[(number)]

The optional number argument is any valid numeric expression.

Return Values

If number is Rnd generates
Less than zero The same number every time, using number as the seed.
Greater than zero The next random number in the sequence.
Equal to zero The most recently generated number.
Not supplied The next random number in the sequence.

Remarks

The Rnd function returns a value less than 1 but greater than or equal to zero.

The value of number determines how Rnd generates a random number:

For any given initial seed, the same number sequence is generated because each successive call to the Rnd function uses the previous number as a seed for the next number in the sequence.

Before calling Rnd, use the Randomize statement without an argument to initialize the random-number generator with a seed based on the system timer.

To produce random integers in a given range, use this formula:

Int((upperbound - lowerbound + 1) * Rnd + lowerbound)

Here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range.

Note: To repeat sequences of random numbers, call Rnd with a negative argument immediately before using Randomize with a numeric argument. Using Randomize with the same value for number does not repeat the previous sequence.

SGN

Returns a Integer indicating the sign of a number.

Syntax

Sgn(number)

The required number argument can be any valid numeric expression.

Return Values

If number is	Sgn returns
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks

The sign of the number argument determines the return value of the Sgn function.

SIN

Returns the sine of an angle.

Syntax

Sin(number)

The required number argument is any valid numeric expression that expresses an angle in radians.

Remarks

The Sin function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

SQR

Returns the square root of a number.

Syntax

Sqr(number)

The required number argument is any valid numeric expression greater than or equal to zero.

TAN

Returns the tangent of an angle.

Syntax

Tan(number)

The required number argument is any valid numeric expression that expresses an angle in radians.

Remarks

Tan takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

IMP

Used to perform a logical implication on two expressions.

Syntax

result = expression1 Imp expression2

The Imp operator syntax has these parts:

Part Description
Result Required; any numeric variable.
expression1 Required; any expression.
expression2 Required; any expression.

Remarks

The following table illustrates how result is determined:

If expression1 is	And expression2 is	result is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

The Imp operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If bit in expression1 is	And bit in expression2 is	The result is
0	0	1
0	1	1
1	0	0
1	1	1

EQV

Used to perform a logical equivalence on two expressions.

Syntax

result = expression1 Eqv expression2

The Eqv operator syntax has these parts:

Part Description
result Required; any numeric variable.
expression1 Required; any expression.
expression2 Required; any expression.

Remarks

If either expression is Null, result is also Null. When neither expression is Null, result is determined according to the following table:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
False	True	False
False	False	True

The Eqv operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If bit in expression1 is	And bit in expression2 is	The result is
0	0	1
0	1	0
1	0	0
1	1	1

XOR

Used to perform a logical exclusion on two expressions.

Syntax

[result =] expression1 Xor expression2

The Xor operator syntax has these parts:

Part Description
result Optional; any numeric variable.
expression1 Required; any expression.
expression2 Required; any expression.

Remarks
If one, and only one, of the expressions evaluates to True, result is True. However, if either expression is Null, result is also Null. When neither expression is Null, result is determined according to the following table:

If expression1 is	And expression2 is	Then result is
True	True	False
True	False	True
False	True	True
False	False	False

The Xor operator performs as both a logical and bitwise operator. A bit-wise comparison of two expressions using exclusive-or logic to form the result, as shown in the following table:

If bit in expression1 is	And bit in expression2 is	Then result is
0	0	0
0	1	1
1	0	1
1	1	0

OR

Used to perform a logical disjunction on two expressions.

Syntax

result = expression1 Or expression2

The Or operator syntax has these parts:

Part Description
result Required; any numeric variable.
expression1 Required; any expression.
expression2 Required; any expression.

Remarks

If either or both expressions evaluate to True, result is True. The following table illustrates how result is determined:

If expression1 is	And expression2 is	Then result is
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

The Or operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If bit in expression1 is	And bit in expression2 is	Then result is
0	0	0
0	1	1
1	0	1
1	1	1

AND

Used to perform a logical conjunction on two expressions.

Syntax

result = expression1 And expression2

The And operator syntax has these parts:

Part Description
result Required; any numeric variable.
expression1 Required; any expression.
expression2 Required; any expression.

Remarks

If both expressions evaluate to True, result is True. If either expression evaluates to False, result is False. The following table illustrates how result is determined:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

The And operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If bit in expression1 is	And bit in expression2 is	The result is
0	0	0
0	1	0
1	0	0
1	1	1

MOD

Used to divide two numbers and return only the remainder.

Syntax

result = number1 Mod number2

The Mod operator syntax has these parts:

Part Description
result Required; any numeric variable.
number1 Required; any numeric expression.
number2 Required; any numeric expression.

Remarks

The modulus, or remainder, operator divides number1 by number2 (rounding floating-point numbers to integers) and returns only the remainder as result. For example, in the following expression

, A (result) equals 5.

A = 19 Mod 6.7

Any fractional portion is truncated. However, if any expression is Null, result is Null. Any expression that is Empty is treated as 0.

NOT

Used to perform logical negation on an expression.

Syntax

result = Not expression

The Not operator syntax has these parts:

Part Description
result Required; any numeric variable.
expression Required; any expression.

Remarks

The following table illustrates how result is determined:

If expression is	Then result is
True	False
False	True
Null	Null

In addition, the Not operator inverts the bit values of any variable and sets the corresponding bit in result according to the following table:

If bit in expression is	Then bit in result is
0	1
1	0