RoboWar Instructions

I. Introduction

Welcome to RoboWar. In this game, you will pit armored robots against each other in gladitorial combat or build your own robots to vie for championship in the arena! Only clever design and skillful implementation will bring your robot to the top. May victory go to the strongest!

Features of RoboWar include animated combat, sound effects, a unique programming language (RoboTalk) with an assembler and interpreter, color graphics on Macs with color monitors, native Power Macintosh support, and a complete robot development factory with a Drafting Board, Hardware Store, Icon Editor, and Recording Studio.

This manual is divided into eight sections. Section I, the introduction, should be self-explanatory. Section II describes the four stations of RoboWar, the Arena, the Drafting Board, the Hardware Store, the Icon Factory, and the Recording Studio, and shows how to use RoboWar to watch predesigned robots fight on the screen. Sections III, IV, and V are for more sophisticated RoboWar users who want to design their own robots. Section III describes the structure of the RoboTalk language and includes a step-by-step tutorial for building a simple robot. Sections IV and V are references detailing the registers and operands used in RoboTalk to control robots. Section VI describes Interrupts, a programming construct that can make your robots significantly more responsive, at the expense of additional programming complexity. Section VII describes the International RoboWar Tournaments in which hackers from around the world duel for glory, and discusses the tournament utility used ot automate such tournaments. Finally, Section VIII contains a rambling history of RoboWar.

If you would like to pit predesigned robots against each other in combat, read the first part of Section II on the Arena. If you would also like to learn to design your own robots, read the rest of Section II, along with Sections III-VI. If you don't like reading manuals, you'll find that running combats is mostly self-explanatory, so you can just start playing with RoboWar right now and come back to skim the later sections when you'd like to build your own robots. If you'd just like to throw away RoboWar and go burn penguins, please see a psychiatrist.

RoboWar is now freeware, not shareware. Individuals are free to use it and give it to their friends. David Harris and Lucas Dixon retain the copyright. Companies interested in including RoboWar on their CDs or other pay sites should contact Lucas at `elmorian@zetnet.co.uk`.

If you have questions or would like to receive mail concerning future tournaments and upgrades, send email to `elmorian@zetnet.co.uk`. Check out the RoboWar Web Page at `http://www.teleport.com/~stiltman/RoboWar/` for the latest RoboWar news and to download recent tournament results. To get on the RoboWar discussion list, send a message reading "subscribe robowar" to majordomo@lists.stanford.edu.

Finally, you mss when you register, you'll get email about tournaments and upgrades. Otherwise, check the RoboWar web page or drop us a postcard for information on upcoming tournaments.

II. RoboWar Control Stations

RoboWar is controlled from five stations: the Arena, the Drafting Board, the Hardware Store, the Icon Factory, and the Recording Studio. One can move between the stations using the first five menu options under the View menu. Each of the stations is described below:

The Arena

The Arena is the primary source of control in RoboWar. When you start the program, you initially are in the Arena. From the arena, you can chose robots to edit or fight and you can run battles.

First you must add robots to the roster in the Arena. Use the New... or Open... commands under the File menu to create and name a new robot or open a predesigned robot. You can have up to six robots open at a time in the Arena. You can select a robot in the roster by clicking on it; a black hilight appears on the robot. The name and icon of the selected robot appears in the bottom right corner of the screen. You can add an additional copy of the selected robot to the roster by choosing Duplicate or remove the selected robot by choosing Close.

Once you have chosen robots to fight, begin the battle by clicking on the Battle button at the bottom of the roster. The robots will be placed in random locations and begin executing their programs until only one robot (or one team) is left alive. The arena displays a number of statistics about the battle. Beside each robot, it lists the robot's current energy and damage ratings, if alive, or the message deceased or buggy, if the robot was blown up or crashed from a defective program, respectively. Robots start at their maximum damage figure and lose damage points as they are hit until they finally explode upon reaching 0 damage. Energy also starts at a maximum value and is used for moving, shooting, maintaining shields, etc. However, it increases at two points per chronon from the robots' generators. Below the roster of robots is a count of how many chronons have elapsed during the battle and how fast the battle is going (chronons per second). A chronon is the elementary unit of time in RoboWar; each robot executes a certain part of its program and moves once per chronon. The chronons per second count will vary based on the speed of the Macintosh you are using, the complexity of the battle (robots and weapons on screen), the state of the speed option, whether sound is being played, and whether the battle is being displayed on screen.

After a robot dies, the RoboWar Mortician reports the time and cause of death. Overloads and buggy robots are reported. If another robot is credited for the kill, the robots name is listed. Kills are only credited when the other robot is still alive and is not an ally of the victim (shooting your teammates or yourself is discouraged). Otherwise, the robot is simply listed as dead.

The previous battle can be replayed by holding down the option key while clicking the Battle button. This is useful for debugging your robot when it does something unexpected. Note that replay still updates the history register each battle, so it is allows you to test your robot starting in exactly the same environment but configured with a different history.

Under the Arena menu are a number of options to configure the battles. The first four are No Team, Team 1, Team 2, and Team 3. These set the team of the selected robot. Robots on the same teams don't register each other on their targeting sensors (the RANGE register, for RoboWar programmers) and can communicate with each other if instructed to do so. Next comes Don't Play Sounds. This turns off all sound effects and beeps. It speeds up the program, saves your sanity, and prevents your boss from discovering the clandestine robot duels occuring at your machine. The next option is the Display item, allowing you to display the entire battle and robot statistics, just the statistics, or nothing at all. The later options are useful if you are running a large tournament in the background and care about speed, not the display. Finally among the display options is the Speed... item. It displays a submenu that lets you control the speed of RoboWar. Fast only updates the screen every other chronon, giving jerkier movement, but speeding up the display somewhat. This option has no benefit if the battle is not being displayed. Normal lets RoboWar run at full speed with complete screen updates. Slow restricts the maximum speed to 30 chronons per second, useful on faster Macs if the battle is too fast to see. Slower restricts the maximum speed to 12 chronons per second, useful for more detailed observation of a battle. Slowest restricts the maximum speed to 2 chronons per second for careful debugging purposes. At the bottom of the Arena menu is the Tournament option, used for running multiple battles. It will be described at the end of this manual.

For programmers, there is an option to use the Debugger. This feature is documented in Section III.

If you just would like to watch combats, you now know everything you need. The rest of this manual describes the other stations used to write robots and the RoboTalk programming language itself.

Drafting Board

The Drafting Board is a text editor and assembler used to write robots. To go to the Drafting Board, select a robot from the roster in the Arena. It may either be a preexisting one you wish to edit or a new robot you wish to program. Then go to the View menu and choose Drafting Board. The window will c

The Drafting Board window is divided into three parts. The square panel on the left with the scroll bar is the text editor window. Use it just like any text editor to type in the robot's program or edit a previous program. Information on how to write a robot will appear in Section III. The bottom right panel shows the name and icon of the robot being edited. Finally, the upper right panel shows statistics on the robot. Under the Program Information heading, the Software Length entry lists the number of lines followed by the number of characters in the current robot's program. The RoboWar editor only allows about 32000 characters in a single robot's program. Next is the Code Length entry which notes the number of instructions in the robot the last time it was assembled using the Compile command. Third is the number of icons in the robot, created in the Icon Factory. Next is the Password Information heading, showing if the robot has a password or not. Finally is a list of dates that the robot was last compiled and at the Drafting Board, Hardware Store, and Icon Factory.

The Drafting Board supports several standard tools under the Edit Menu: Undo, Cut, Copy, Paste, Clear, and some search and replace options. They are essentially self-explanatory.

Under the View menu is the Compile command. It is used to translate the RoboTalk code typed into the text editor of the Drafting Board into a special internal representation used for efficient interpreting. Every time you edit a robot's program, you must compile it again to reflect the changes in the robot's actual behavior. If the compiler detects any syntax errors in the robot (label names that don't exist, misspelled operands, etc.), it will display a message and hilight the word that is in error. Also, if you have any text selected, the compiler will inform you of the number of instructions in the highlighted region (including implicit RECALL statements).

Note that you must register your copy of RoboWar to compile robots of more than 100 instructions. This should allow you to write relatively simple robots and learn your way around the program. However, if you become a serious hacker, you really ought to send in your registration fee. Moreover, as a registered user, you'll get information about future tournaments and major new versions of the program. I intentionally kept the registration fee very low so that everyone could afford the program; if $15 is a serious drain on your finances, let me know and we'll work something out.

Finally, under the View menu, is the Set Password command. A robot's password can be set from any of the RoboWar stations, including the Arena. Enter the password into the dialog box, or hit return if you don't want a password. If you do want a password, the computer will prompt you to enter the password again to make sure you didn't accidentally hit a wrong key. Once a robot has a password, you must enter the password to go to the Drafting Board, Hardware Store, or Icon Factory for the first time after the robot is opened. This means that somebody else could try your robot in the Arena and watch it fight, but couldn't look at or edit the code without knowing the password. Please note: robots with passwords are stored in scrambled form. If anything goes wrong, it is very easy to lose your robot's code forever. Therefore, placing passwords on your robot can be risky. I would recommend that you don't put passwords on robots until after they are completely finished, and that you always keep a backup of the robot without a password on your own disk somewhere safe. In the immortal Latin tradition, "Cave passverbum."

Hardware Store

The Hardware Store allows you to customize a robot's equipment and weapons systems. To visit the Hardware Store, select a robot from the roster in the Arena, then choose Hardware Store from the View menu.

In the Hardware Store, you may buy up to nine points of equipment for your robot. Points can be spent on the robot's maximum energy, damage, and shield values, the robot's processor speed, the type of bullets the robot has, and the various additional weapon systems you wish to install on the robot.

The maximum energy value is the amount of energy the robot starts with at the beginning of a combat and also the maximum amount the robot could ever reach by letting power recharge from its power supply. The choices are: 150 (3 points), 100 (2 points), 60 (1 point), and 40 (no cost).

The maximum damage value is the amount of damage a robot can sustain before being destroyed. The choices are 150 (3 points), 100 (2 points), 60 (1 point), or 30 (no cost).

The maximum shield value is the greatest amount of power that can be stored in a shield at normal drain rates. Up to this amount of power may be placed in a shield; the power in the shield will decrease at half a point per chronon. Quantities of energy greater than the shield max may be stored in a robot's shields, but the power will decay at two points per chronon. The options for shield max are 100 (3 points), 50 (2 points), 25 (1 point), and 0 (no cost).

The processor speed indicates how many instructions a robot executes per chronon. The options are 50 (4 points), 30 (3 points), 15 (2 points), 10 (1 point), and 5 (no cost).

The Bullets choice determines what kind of ammunition is available for the robot's built-in gun mount. The options are explosive (2 points), normal (1 point), and rubber (no cost). Finally is a list of additional weapons that can be mounted on a robot: Missiles, TacNukes (tactical nuclear devices), Hellbores, Stunners, and Mines. The weapons is described below:

Every weapon has a certain amount of energy placed in it at launch that will be referred to as "energy" in this paragraph. All bullets move at speed 12. Explosive bullets set off a small nuclear explosion on impact. The explosion radius grows to 36 pixels over three chronons, then detonates, causing 1.5*energy to any robots in the blast radius (possibly including the robot who fired)! Normal bullets just cause damage equal to energy when they hit. Rubber bullets cause damage equal to 0.5*energy when they hit. Missiles move at speed 5 and cause damage equal to 1.5*energy if they hit. TacNukes don't move; they just grow to a radius of 50 pixels over ten chronons, then detonate like explosive bullets for 1.5*energy to any targets foolish enough to be present. A wise robot using TacNukes will rapidly move away after depositing the nuke as to escape the blast radius. Hellbores move at speed equal to the energy invested (between 4 and 20) and knock down the opponents shield if they hit but cause no other damage. Stunners are stasis capsules fired at hostile robots. They move at speed 14 and put anything they hit into stasis, a state in which a robot does not move, recover energy, execute instructions, or lose shield strength to natural decay. For every four points of energy invested, the target will remain in stasis for one chronon. Finally, mines are dropped in place and are inactive for 5 chronons before becomming armed. Once armed, they do damage equal to two times (energy-5) to any robot that wanders over one.

An interesting twist on RoboWar is to compete with friends building robots that use very few hardware points and/or instructions. For instance, you might limit everyone to four points and 500 instructions and see what robots you can create. An option under the Arena menu allows you to specify the maximum number of hardware points for a particular battle. Two suggestions are Titan robots (21 points) and Little League robots (2 points). Some RoboWar tournaments feature Titan and/or Little League categories, depending on popular demand.

On the right side of the window are some other configurations for the robot. There is a choice of what kind of indicator to use to mark the robot's turret: either a line from the robot's center out, a dot in the direction of the turret, or no marker at all. This is a purely cosmetic option. Finally there is a check box labeled No Negative Energy. If checked, a robot will never overload its power supply and go to negative energy. The advantage is that the robot will never run out of energy and freeze in place until recovering; the disadvantage is that a robot might not have enough energy to complete a shot or some movement.

Icon Factory

The Icon Factory is a station in which you design icons for your robots. As with the other stations, first you must select a robot from the roster. Then, choose Icon Factory from the View menu to go to the station.

The Icon Factory has three panels. The large square on the left is the magnified view of the icon being edited; the ten squares on the right are the robot's various icons; and the box in the bottom right shows the robot being edited.

To draw an icon, select it at the right. The icon labeled standard (icon 0) is the robot's normal icon that will appear in the roster.

It is used when the robot has no shields enabled and hasn't explicitly been instructed to use another icon. Icon 1 (shields) is the icon the robot uses by default when shields are enabled. Icons 2-8 are user-defined pictures that can be selected by the robot during combat. The check-boxes next to icons 1 through 5 can be used to automate icon changing: When the Shield check box is on, whenever your robot's shield is above 0 icon1 will be used for your robot; The Death Check box will make your robot set it's, when your robot dies, to icon2; Collide will set your robot's icon every time your robot is in a collision; Block will set your robot's icon whenever your robot stops some damage via it's shields; Hit will make your robot change to icon5 whenever it is hit and damage is taken.

Click on the icon you wish to design or edit. On the left, use the 32x32 grid to draw the icon. The crosshairs function much like the pencil in a paint program. The four arrows on the edges of the grid allow you to scroll the picture left, right, up, or down. The other tools allow you to rotate and flip your icons horizontally or vertically. Lucas Dixon added really nifty color icon support to RoboWar 4.3.

You can also use the Edit menu to cut, paste, copy, or clear icons. This is especially useful because it allows you to copy icons or pictures generated by other programs (e.g. SuperUltraMagnificentWayNiftyAndExpensivePaint) and paste them into your robot.

Once you are satisfied with the icons you have created, you can return to the Arena from the View menu.

Recording Studio

The Recording Studio is a station in which you digitize sounds for your robots. As with the other stations, first you must select a robot from the roster. Then, choose Recording Studio from the View menu to go to the station.

Each robot can have up to ten sounds defined. These sounds are indicated on the right portion of the screen. The first four sounds have special meanings. Sound #0 is the Death sound, played when a robot explodes. Sound #1 is the Collision sound, played when the robot is in a collision. Sound #2 is the Shield Hit sound, played when the robot is hit by a weapon that fails to penetrate shields. Finally, Sound #3 is the Robot Hit sound, played when the robot is hit and takes damage.

To record or play a sound, select the appropriate rectange. If a sound already is defined, the rectangle will be filled with gray. On the left side of the screen will be icons to record a new sound or play a sound if it is already defined. You can also adjust the degree of compression, from none to 6-fold. Compressed voice is scarcely different in sound than uncompressed voice, so you can save a great deal of disk space through suitable compression. Musical effects tend to suffer more from compression. Individual sounds recorded in RoboWar are automatically limited to 40K in length to prevent extensive disk-hogging.

As usual, you can cut, paste, copy, or clear sounds via the Edit menu. Thus, you can copy sounds from other programs (for instance, sounds that you have recorded and touched up in a commercial recording program) and paste them into RoboWar.

Be sparing and careful with your use of sounds. Heavy use will slow the program and hog lots of disk space; for instance, if every robot in a tournament of 30 entries had ten very short sounds (6K each), the robots from the tournament would occupy more space than RoboWar itself and all the robots from Tournaments I through V combined! Moreover, if you use too may sounds, you will quickly fill up your memory.

Apple's Sound Manager calls seem to work differently on every machine around, so sometimes sounds don't work very well or simply crash the machine. I am still trying to solve the problems, but a workaround is to check "Don't Play Sounds" under the Arena menu if sounds crash RoboWar on your machine.

Once you are satisfied with the sounds you have created, you can return to the Arena from the View menu.

III: An Introduction to RoboTalk

RoboTalk is the language in which robots are programmed for RoboWar. The language is based on Reverse Polish notation (RPN), similar to the HP calculators. This section introduces a number of important RoboTalk concepts: the Stack, operands and operators, comments, tokens, delimiters, and programs, numbers, label definitions, labels, and variables. It also describes the hardware interactions that robots have, including the effects of energy, shields, weapons, and collisions. The section concludes with a step-by-step tutorial of creating your own robot.

Some examples are based on a simple robot, ShotBot. ShotBot's program is listed below:

```
# ShotBot
# Written 1/3/90 by David Harris

Main:
    RANGE 0 > FireSub RotateSub IFE
    Main JUMP

FireSub:
    20 FIRE' STORE
    RETURN

RotateSub:
    5 AIM + AIM' STORE
    RETURN
```

The Stack

The stack is the basis of all RoboTalk instructions. It works like a stack of paper: a new sheet may be placed (pushed) onto the top of the stack or removed (popped) from it. However, pieces cannot be taken from or inserted into the middle. All operands, such as numbers, variable names, and labels are pushed onto the stack. Operators may pop information off the top of the stack, act on it, and push any result back onto the stack.

The most recently pushed operand on the stack is called the top of the stack. A new operand pushed onto the stack is placed above the old top and becomes the new top of the stack. An operand is popped off the top of the stack, leaving the operand beneath it as the new top. If more than 100 operands are on the stack at once (an unlikely occurrence unless the robot has a bug), the robot has a Stack Overflow error. If one tries to pop a number off of the stack when it is entirely empty, a Stack Underflow error is reported.

Tokens, Delimiters, and Programs

A token is a group of characters, usually a number or word. Every element of RoboTalk: numbers, label definitions, labels, variables, and operands, are really tokens. Each token must be separated from the next by at least one delimiter.

A delimiter is a symbol that separates two tokens. The most common delimiters are spaces and new lines. For example, in the line "`Main JUMP`" the space separating Main and Jump is a delimiter. Other valid delimiters include the tabs, semicolons, and commas. RoboTalk does not care which delimiters are used where. Thus ShotBot could be rewritten as:

```
Main:,Range;0 >;FireSub
RotateSub,IFE,Main;JUMP;FireSub:;;;20 fire'    STORE,RETURN,RotateSub:
  5 AIM + ,AIM' STORE;RETURN
```

However, this is very unclear for the reader. Therefore, a style similar to that shown in some of the example robots, with spaces and new lines used for delimiters, and indentation following label definitions, is recommended.

A program is a series of tokens that work together to control a robot. Each robot's software is a single program. The segments

branched to by IF and CALL operators are called subroutines. Each of the types of tokens is described below.

Comments

Comments are messages that a human may use to help understand a program, but that are ignored by RoboTalk. For example, in ShotBot, described in Part I, the line "`# Written 1/3/90 by David Harris`" is a comment. It reports to the user information about the robot's author, but does not generate any code that RoboTalk interprets.

Comments come in two varieties for convenience. The first variety begins with a # sign. This comment means that the rest of the line is a comment and should be ignored. The comment might come at the beginning of the line, or might follow some real code, as in the line:

```
AIM 5 + AIM' STORE  # Rotate Turret.
```

The other form of comment is marked by the { and } symbols (open and close braces.) The open brace indicates a start of the comment. The comment is ended by the close brace. There must be a close brace for every open brace. However, comments may be nested; that is, a pair of open and close braces may appear between another set of braces. The following program shows an example of comments marked by braces:

```
# An example robot

{   This is the beginning of a comment.
    It continues on the next line.
        {This comment is nested within the outer one
    } The previous comment is closed.

main: # this is not a label definition; it is enclosed in a comment
}

main:  # this is a valid label definition
    main jump
```

Label Definitions

Label definitions mark a point in the program so that jumps and branches may go to that point. A label definition consists of a word followed by a colon. Label definitions do not generate any RoboTalk code. However, they are used to mark the destination of labels in other parts of the program.

Label definitions should not have the same names as variables or operators. Also, there may not be two label definitions with the same names in a single program.

Operands and Operators

All tokens other than comments and label definitions are either operands or operators. An operand is a number, label, or variable name that is pushed onto the stack. An operator is a command that acts on the stack. The various operands and operators are described below.

Numbers

A number is just that: a collection of digits. Numbers are always pushed onto the stack when they are encountered. Numbers may range from -19999 to 19999. They may have a plus or minus sign in front of them; however, there must be no delimiters between the sign and the digits. (If there were, RoboTalk would interpret it as an operand, either plus or minus, followed by a positive number.) If an arithmetic operation produces a number out of this range, for instance, from executing "200 200 *", the robot bombs with an error message "Number out of bounds."

Labels

Labels are used with operators such as IF, IFE, IFG, IFEG, JUMP, and CALL. They are coded as the location in the program to jump to, and thus are pushed onto the top of the stack when encountered.

Variables

Variables, also known as robot registers, contain information from the robot. They include the range to the nearest robot in the sights, a robot's current X and Y position, energy, etc. A complete list of variables and their functions appears in Part V.

Variables may appear in two forms in a program: unquoted and quoted. An unquoted variable really refers to the contents, or value, of that variable, while a quoted variable indicates the variable name itself. Quoted variables (written as the variable name followed immediately by a single quotation mark e.g. AIM') are used with the STORE operator, as they are the location in which a number should be stored. Quoted variables must be used with every STORE operator, and should not be used with any other expression. For example, although "`5 AIM' +`" will successfully assemble, the results will be meaningless and will probably cause the robot to do strange things, because the number 5 is being added to the name of the variable AIM, not the contents of AIM.

Advanced programmers should understand that an unquoted variable translates internally into a quoted variable followed by a RECALL statement. Thus, it takes two cycles to access an unquoted variable, as opposed to the one cycle for other instructions. The RECALL statment has been added explicitly to RoboWar 4.4, so a programmer could write "AIM' RECALL" instead of just AIM. Either form takes

two cycles.

Operators

Operators may pop information off of a stack, act on it, and push information back onto the stack. A complete list of operators appears in Part IV. They include mathematical functions like +, -, and *, stack manipulation functions like DROP and SWAP, and branching functions like JUMP and CALL.

Advanced programmers should note that all branches that may be returned from, namely, those generated by IF, IFE, and CALL, push the return address onto the stack. Thus if a subroutine were to be written that would act on information on the stack, it would first have to save the return address before acting. Then, it would have to restore the return address before making a RETURN statement. The following code, which invokes a subroutine to double the number on the stack, demonstrates this technique:

```
# Demonstration Robot

Main:
    5              # the number to double
    DoubleSub CALL # double it
    DROP           # discard it
    Main JUMP      # repeat forever...

DoubleSub:
    r' STORE       # store return address in variable r
    2 *            # multiply top number on stack by 2
    r              # restore return address to stack
    RETURN         # and return
```

An alternative (better) method of writing DoubleSub would use stack manipulation commands, saving an instruction:

```
DoubleSub:
    SWAP     # swap return address and number to multiply
    2 *      # multiply top number on stack by 2
    SWAP     # swap return address back to top of stack
    RETURN   # and return
```

The first DoubleSub routine executes in 7 cycles (recalling that an unquoted variable requires two cycles), while the second routine executes in only 5 cycles.

Note that the IFG and IFEG branching commands do not leave any return address on the stack, so they are useful for transfering control permanently without requiring a DROP to clean up the stack.

Special Operators

The ICONx and SNDx (where x is a number from 0 to 9) operators are used for special effects. Therefore, they take zero cycles to execute. Icons are defined in the Icon Factory, while sounds are set in the Recording Studio. Be sparing with the use of sound or you will create disk-hogging, battle-slowing behemoths that nobody else wants on their hard disk!

DEBUGGER is another special command. If the robot debugger is enabled (by selecting the robot and chosing Use Debugger from the Arena menu), the DEBUGGER command causes a robot to end all processing for the chronon (like the SYNC command) and enter the debugger, in which one can examine robot registers and single-step code. If the robot debugger is not enabled, the DEBUGGER command is ignored and takes no time to execute, so a robot can be properly tested.

Interrupts

Interrupts are a new feature of RoboWar, as of version 3.0. They are an advanced programming feature that novices may safely ignore, yet they present the opportunity for experts to write much more efficient (and hence deadly) code.

Traditional robots use the "polling" style of programming, checking for events in a main loop. For example, a typical moving robot would have to check each time through the main loop if it is too close to a wall, if it is in a collision, if any projectiles are coming toward it, and if it sees any enemies to shoot at. Worse yet, a team robot might have to check if its partner had sent any messages recently. All of this checking for conditions that are important but may rarely occur requires lots of processor time.

Interrupts offer a new solution. Instead of polling for events, an interrupt-driven robot can request that normal execution be "interrupted" when a special situation is detected. The interrupt causes a robot to jump to a special procedure defined to handle the event.

See Section VI for more information about interrupt-based programming.

Assembler Directives

Assembler directives control how the assembler processes your program. They are only needed by advanced RoboTalk hackers, so skip this section if it makes no sense. Aseembler directives begin with #!!, so they look similar to comments. Assembler directives have no effect on how your robot operates, only how the assember reads your code. Two assembler directives are currently supported:

```
#!! NoStoWarnings
#!! StoWarnings
```

These turn off and on the warning about STORE with unquoted variable. An application of these directives is for programmers who want to fire several bullets in rapid succession (for example to spray the arena). One way of doing this is:

```
 0 Aim' STORE  10 Fire' STORE
30 Aim' STORE  10 Fire' STORE
60 Aim' STORE  10 Fire' STORE
90 Aim' STORE  10 Fire' STORE
```

This takes a total of 24 instructions, so the bullets may be fired over several chronons. A robot may need to fire in a single chronon, so advanced programmers use the technique:

```
10 Fire' 90 Aim'
10 Fire' 60 Aim'
10 Fire' 30 Aim'
10 Fire'  0 Aim'
SYNC
STORE STORE STORE STORE
STORE STORE STORE STORE
```

This technique is legal and dispatches all the shots in 8 instructions starting at the beginning of a chronon. However, the STORE operators are now separated from their operands. In this case, the code is correct, but often when a beginner separates STORE from the operands, it is a mistake so the STORE without quoted variable warning is issued.

The NoStoWarnings and StoWarnings directives are used to show you know what you are doing and prevent the assembler from giving annoying warnings. Remember to turn StoWarnings back on after the STOREs to catch any mistakes you might have in your code (unless you are a perfect programmer who makes no mistakes).

```
10 Fire' 90 Aim'
10 Fire' 60 Aim'
10 Fire' 30 Aim'
10 Fire'  0 Aim'
SYNC
#!! NoStoWarnings
STORE STORE STORE STORE
STORE STORE STORE STORE
#!! StoWarnings
```

Maximum Size Limitations

RoboWar places certain limitations on the size of robots. Each robot may have no more than 5000 instructions not counting icon and sound tokens. Also, there may be no more than 400 labels and each label must be less than 20 characters long.

Sample Program

Now that we have learned all the elements of a program, let us see how they act on the stack in a simple robot. The following robot remains stationary, rotating its turret:

```
# SimpleRobot

Main:
    Aim 5 +             # Rotate 5 degrees
    Aim' STORE
    Main JUMP           # Repeat
```

Suppose that the variable Aim currently contained 90 degrees, due east. Let us trace the stack through each instruction:

Instruction: # SimpleRobot
Type: Comment
Stack: Empty

Instruction: Main:
Type: Label Definition
Stack: Empty

Instruction: Aim
Type: Unquoted variable = 90
Stack: 90 <-- Top of Stack

Instruction: 5
Type: Number
Stack: 5 <-- Top of Stack
90

Instruction: +
Type: Operator

Stack: 95 <-- Top of Stack

Instruction: # Rotate 5 Degrees
Type: Comment
Stack: 95 <-- Top of Stack

Instruction: Aim'
Type: Quoted Variable
Stack: Aim' <-- Top of Stack
95

Instruction: STORE
Type: Operator
Stack: Empty

Instruction: Main
Type: Label
Stack: Main <-- Top of Stack

Instruction: JUMP
Type: Operator
Stack: Empty

The program repeats this cycle, incrementing Aim by 5 degrees each time through the loop.

Hardware Interactions

The robot interacts with his environment by reading and writing variables. For example, a robot could check his X position by reading the variable X, or could fire a bullet by writing some amount of energy to the Fire variable. Bullets, missiles, and TacNukes were described in Part II, and in more detail in Part IV. This section gives an overview of the robot's energy and damage statistics, as well as the effect of collisions with other robots and walls.

The robot's energy, listed in the upper right box during combat, is the total amount of power available to the robot. It is used for acceleration, maintaining shields, and shooting weapons. Each chronon the energy recharges by two points.

Large amounts of energy consumption may place a robot at negative energy. On any chronon in which a robot begins with negative energy, it cannot move or interpret any instructions. It is just a sitting duck until the energy returns to the realm of positive numbers.

Finally, if energy ever drops below -200 (from a massive expenditure in a single chronon) or a robot attempts to use more than 600 energy in a single instruction, the robot's power supply melts down and the robot explodes. This is generally a bad thing.

The robot's damage, also listed in the upper right box during combat, is how much damage may still be taken before the robot is destroyed. Damage does not regenerate; once a robot takes damage, the damage is present until a new battle starts.

Collisions are the bane of any moving robot. Two types of collisions exist: collisions with walls and collisions with other robots. Collisions with walls are the most dangerous, but also the most avoidable. If a robot hits a wall, it takes five points of damage per chronon until it moves back into the main arena. However, checking the X and Y positions of a robot, and adjusting velocity if the robot nears a wall, should prevent this kind of collision.

Collisions with other robots are sensed in the Collision variable. It returns a 1 if read while the robot has hit another. Each chronon that the robots are touching causes 1 point of damage to both combatants. Furthermore, the robots cannot move through each other, so they may end up locked together until one or the other perishes. However, a smart moving robot will check the collision variable frequently. If he has collided, he may jump to some code that locates and destroys his opponent.

Debugger

Starting with version 2.3, RoboWar has a debugger that allows you to trace your RoboTalk programs, examine registers, and observe the contents of the stack.

To use the debugger, click on an open robot in the Arena. Choose the Use Debugger option from under the Arena menu. The Use Debugger menu item will be checked. To use the debugger on a different robot, select that robot instead and choose Use Debugger again. To turn off the debugger, click at an empty line of the Arena to deselect all robots, then choose Use Debugger and notice that the check beside the menu item is removed. If you are using the debugger on a robot with a password, you may be prompted to enter the password.

When you run a battle with the debugger enabled, the right portion of the Arena window will display information about the robot being debugged. Two boxes near the top display the program and the top five elements of the stack. Below is a list of the robot's variables and their current values.

Four buttons control execution. When the battle begins, the robot will be paused. The Go button begins free-running execution, just as in a normal battle. Chronon executes for one chronon, then pauses again. Step executes a single instruction at a time. When the robot is free-running, clicking the Pause button pauses at the end of the chronon. If a DEBUG instruction is encountered in program execution, the robot will also pause at the end of that chronon. If a robot crashes, the program window will display the instruction that caused the crash.

Keyboard shortcuts are available for the debugger: G, P, C, and S correspond to Go, Pause, Chronon, and Step, respectively.

In order to help with debugging, one may pick up any robot on the screen by clicking on it and may drag the robot to a new location. Of course this is for testing only; I would require a bribe of an amount unable to fit in a 10 byte IEEE standard floating point format to bias tournaments in this fashion.

Be aware that unquoted variable references are internally represented as a variable name (quoted variable) followed by a RECALL operation. Also note that the debugger does not show the quotes on variable names. Thus, the RoboTalk code in listing A would be displayed as listing B in the debugger:

Listing A: `aim 5 + aim' store`
Listing B: `AIM RECALL 5 + AIM STORE`

The debugger is missing a few features due to viewing space limitations. It does not display labels or label definitions in the code. It does not display user-defined variables; however, you can check the value of a variable you are accessing by simply single-stepping through that section of code and observing what is placed on the stack. There is also no way to scroll through the code or stack.

Finally, you should be warned that robots do not behave 100% identically when the debugger is turned on. Specifically, when the debugger is on, situations involving negative energy and stunners may affect the robot slightly differently and the robot's code is executed after all other robots, possibly changing the outcome of timing critical events like collisions.

Tutorial

Now that we know the basics of RoboWar, let us design a robot. We'll create a robot that seeks a position along the top wall, then scans back and forth, shooting at any targets it sees.

First, we need to create the robot. Run RoboWar or make sure you are in the Arena if you are already running RoboWar and choose New Robot... from the File menu. Give the new robot a name, say TopBot.

Notice that the robot is hilighted in the roster and that it appears in the bottom right corner as the selected robot. Now we must go to the drafting board and write the robot's code. Go to the View menu and choose Drafting Board.

The Drafting Board should have no code in it since we haven't started the robot. Type in the following program:

```
# TopBot
# Written by <YourName> on <Date>

Start:
      0 aim' store                # point turret up
      -4 speedy' store            # start moving up

SeekTop:
      range 0 > KillTarget if     # shoot at target
      y 30 < ReachedTop if        # at top yet?
      SeekTop jump                # repeat

ReachedTop:
      drop                        # return address
      0 speedy' store             # stop moving
      90 aim' store               # point gun right

MainLoop:
      range 0 > KillTarget if     # shoot at target
      aim 5 + aim' store          # rotate turret
      aim 270 > RotateBack if     # half circle
      MainLoop jump               # repeat forever

KillTarget:
      20 fire' store              # fire a bullet
      return                      # and continue

RotateBack:
      90 aim' store               # point gun right
      return
```

Check over the program to make sure you typed it properly. Then choose Compile from the View menu to compile the program. If Compile tells you that you have a bug in your program, check it again against the listing above and fix your mistake. When the program compiles properly, look at the statistics on the right. It should say that your program is 57 instructions long.

Now, let us go to the Hardware Store to add some features to the robot. Choose Hardware Store from the View menu. Looking at the window, note that, by default, the robot has a shield max of 50. However, since TopBot doesn't use shields, we don't need any shield max. Set it to 0 by clicking on the circle labeled Zilch under the Shield Max heading. That gives us six points so far on hardware. We can get three more.

Choose explosive bullets from the Bullets box. Then raise the Damage Max and the Energy Max to 150. Looking at the info on the right again, we see we now have spent all nine points. Return to the Arena by choosing Arena from the View menu.

Let us put in another copy of TopBot so that the two robots can battle. Be sure that TopBot is selected, then choose Duplicate from the File menu. This will put a second copy of TopBot in the Arena. Now click on the button labeled Battle. Watch the results! Try a few battles, then add some other pre-designed robots if any came with your copy of RoboWar. What are TopBot's strengths? Weaknesses? How can its design be improved?

Now you are ready to design and construct your own robots. Look through the next two sections on operators and registers to learn about the full spectrum of robot capabilities. Then see what you can invent! I would love to see your best creations or watch them compete in a RoboWar tournament.

IV: Operators

This section lists each operator. It gives the name and the effect it has on the stack or robot state. The operands are as follows:

+	-	*	/	=
!	<	<	ABS	AND
ARCCOS	ARCSIN	ARCTAN	BEEP	CALL
CHS	COS	DEBUGGER	DIST	DROP
DROPALL	DUPLICATE	FLUSHINT	ICON0-9	IF
IFE	IFEG	IFG	INTOFF	INTON
JUMP	MAX	MIN	MOD	NOP
NOT	OR	PRINT	RETURN	ROLL
RTI	SETINT	SETPARAM	SIN	SND0-9
STORE	SQRT	SWAP	SYNC	TAN
VSTORE	VRECALL	XOR		

A number of these commands have abbreviations: DUP is an acceptable substitute for DUPLICATE, STO for STORE, DEBUG for DEBUGGER, COSINE for COS, SINE for SIN, TANGENT for TAN, and EOR for XOR. RETURN and JUMP really perform the same function, as noted below; hence, in the debugger, both instructions are listed as JUMP. In the debugger, the special (RECALL) and (END) operators may appear; these are inserted by the Assembler and not used directly by the programmer.

+
Adds the top two numbers on the stack, removes them, and replaces them with the result.
Ex: "4 5 +" leaves 9 on the top of the stack.

-
Subtracts the top number from the second number on the stack, removes them, and replaces them with the result.
Ex: "9 3 -" leaves 6 on the top of the stack.

*
Multiplies the top two numbers on the stack, removes them, and replaces them with the result.
Ex: "2 4 *" leaves 8 on the top of the stack.

/
Divides the second number by the top number on the stack, removes them, and replaces them with the result.
Ex: "22 3 /" leaves 7 on the top of the stack. (7.3333 is truncated to 7)

=
Checks if the top two numbers on the stack are equal, then removes them. If they are equal, it places a 1 on the stack, otherwise it pushes a 0.
Ex: "2 2 =" leaves a 1 on the stack.

!
Checks if the top two numbers on the stack are not equal, then removes them. If they are not equal, it places a 1 on the stack, otherwise it pushes a 0. (The symbol ! comes from the logical NOT command in the language C.)
Ex: "5 5 !" leaves a 0 on the stack.

>
Checks if the second number on the stack exceeds the top number, then removes them. If the second number is greater, it places a 1 on the stack, otherwise it pushes a 0.
Ex: "5 4 >" leaves a 1 on the stack.

<
Checks if the second number on the stack is less than the top number, then removes them. If the second number is less, it places a 1 on the stack, otherwise it pushes a 0.
Ex: "7 3 <" leaves a 0 on the stack.

ABS
Absolute Value. Removes the top argument and returns its absolute value.
EX: "-8 ABS" leaves 8 on the stack.

AND
Checks if the top two numbers on the stack are both not zero, then removes them. If they are both not zero, it places a 1 on the stack, otherwise it pushes a 0.
EX: "2 3 AND" leaves a 1 on the top of the stack.

ARCCOS

Inverse Cosine. Computes the principal inverse cosine of the top number divided by the second number. Returns the result in degrees in the range of 0 to 180 (just like the C library routine). Note that you may have to convert the result to the appropriate coordinate system before using it to set your AIM register or such.
EX: "1000 707 ARCCOS" leaves 45 on the stack.

ARCSIN
Inverse Sine. Computes the principal inverse sine of the top number divided by the second number. Returns the result in degrees in the range of -90 to 90 (just like the C library routine). Note that you may have to convert the result to the appropriate coordinate system before using it to set your AIM register or such.
EX: "1000 707 ARCSIN" leaves 45 on the stack.

ARCTAN
Inverse Tangent. Computes the inverse tangent of the ratio of the top two numbers. The y value must be the top operand; the x value must be the second operand. ARCTAN removes the top two operands and returns the arctangent of y/x. The result is in degrees between 0 and 359, with 0 degrees pointing up, just as with AIM angles. Also remember that the positive x values are on the right side of the arena, while positive y values are on the bottom of the arena.
EX: "-5 0 ARCTAN" leaves 270 on the stack.

BEEP
Beeps once. Most useful in debugging a robot.
EX: "BEEP" leaves nothing on the stack.

CALL
Jumps to the instruction number specified by the top element of the stack, removes the top element, places the return address (the instruction number previously being executed) on the top of the stack, and resumes execution at the new instruction. Very similar to JUMP, but leaves the return address on the stack.

CHS
CHange Sign. Multiplies the top operand on the stack by -1, removes it, and returns the result on the stack.
EX: "3 CHS" leaves -3 on the stack.

COS or COSINE
Cosine function. The top argument should be the hypotenuse and the second argument should be the angle. COS removes the top two arguments and returns the hypotenuse times the cosine of the angle, truncated to an integer value. The angle must be between 0 and 359 or the result is undefined. Also, note that while 0 degrees is pointing straight up on the turret, the cosine of 90 degrees is still 0 (e.g. COS uses the standard trig coordinate system).
EX: "30 100 COS" leaves 86 (=100*cos(30) on the stack.

DEBUG/DEBUGGER
A hook for debugging. If the robot is selected for debugging (i.e. has the bug icon beside it in the Arena display), tthis command acts as a sync instruction, pausing until the end of the chronon (because the debugger cannot turn on in the middle of a chronon). Execution pauses immediately after the debugger instruction and one can use the debugger facilities. If the robot is not selected for debugging, nothing happens and no time is used (so that one can run accurate timing on a robot without having to remove DEBUG statements).
EX: "DEBUGGER" pauses until the end of chronon and enters the debugger if debugger is on

DIST
Calculate the distance from the origin to some point (x,y) = Sqrt(x*x+y*y). The top two elements of the stack are x and y; they are removed and replaced with the distance.
EX: "10 10 DIST" leaves 14 on the stack.

DROP
Drops the top element from the stack.
EX: "5 DROP" leaves nothing on the stack.

DROPALL
This command drops everything off of the stack. It is useful for interrupt routines that may not return to where they were called, thus leaving an unknown amount of junk on the stack. It is also useful for sloppy programmers who let junk accumulate on the stack and run into stack overflows after lots of chronons.
EX: "87 42 99 -32 DROPALL" leaves absolutely nothing on the stack.

DUPLICATE or DUP (either token is allowed)
Duplicates the number on the top of the stack.
EX: "5 DUP" leaves 5 on the top of the stack and 5 in the second position.

FLUSHINT
Removes all pending interrupts from the interrupt queue.
EX: "FLUSHINT" does nothing to the stack.

ICON0-9
Sets the robot's icon. ICON0 is special; it causes the robot to display icon #0 if shields are off, but icon #1 if shields are on. Other ICONx commands display icon #x. Takes zero chronons to execute. The various icon commands are used for icon animation.
EX: "ICON3" does nothing to the stack but makes the robot display icon #3.

IF
Checks the second operand on the stack. If it is not zero then it leaves the return address on the stack and jumps to the label specified on the top of the stack. In any case, it removes the second operand and the destination label from the stack.

EX: “1 MySub IF” jumps to the subroutine MySub and leaves the return address on the stack.

IFE
Stands for IF-THEN-ELSE. Checks the third operand on the stack. If it is not zero than it leaves the return address on the stack and jumps to the label specified in the second position on the stack. If it is zero then it leaves the return address on the stack and jumps to the label specified on the top of the stack. In any case it removes the first, second, and third elements from the stack.
EX: “0 SubA SubB IFE” jumps to the subroutine SubB and leaves the return address on the stack.

IFG
Stands for IF-GOTO. Checks the second operand on the stack. If it is not zero then it jumps to the label specified on the top of the stack wi
EX: “RANGE TARGETING IFG” jumps to the label TARGETING if a hostile robot is in the line of fire and leaves nothing on the stack.

IFEG
Stands for IF-THEN-ELSE-GOTO. Checks the third operand on the stack. If it is not zero than it jumps to the label specified in the second position on the stack. If it is zero then it jumps to the label specified on the top of the stack. Unlike IFE, it leaves no return address. In any case it removes the first, second, and third elements from the stack.
EX: “0 SubA SubB IFEG” jumps to the subroutine SubB, leaving nothing on the stack

INTOFF
Turns interrupts off until next INTON command.
EX: “INTOFF” does nothing to the stack.

INTON
Turns interrupts on. Must be used near start of any robot that uses interrupts.
EX: “INTON” does nothing to the stack.

JUMP
Jumps to the instruction number specified by the top element of the stack, removes the top element, and resumes execution at the new instruction. This is the same operation as RETURN.

MAX
Leaves only the greater of the top two numbers on the stack.
EX: “4 8 MAX” leaves 8 on the stack.

MIN
Leaves only the lesser of the top two numbers on the stack.
EX: “4 8 MIN” leaves 4 on the stack.

MOD
Performs a modulus operation (remainder of integer division) on the top two elements of the stack. Removes them and returns the result on the stack. Note that this follows the same rules as the modulus operation in C; for example, -69 mod 360 = -69.
EX: “10 3 MOD” leaves 10 mod 3 = 10-3*Trunc(10/3) = 1 on the stack.

NOP
No OPeration. Does nothing whatsoever, except take up time and space. May be used when some timing loop is necessary.
EX: “NOP” leaves nothing on the stack.

NOT
Logical Not. Checks top operand, removes it. Returns 1 if it was 0, 0 otherwise.
EX: “4 NOT” leaves 0 on the stack.

OR
Checks if either of the top two numbers on the stack is not zero, then removes them. If either is not zero, it places a 1 on the stack, otherwise it pushes a 0.
EX: “0 4 OR” leaves a 1 on the top of the stack.

PRINT
Displays the top element of the stack in a dialog box. Takes zero chronons to execute. Intended for debugging purposes only.
EX: “5 PRINT” leaves 5 on the stack and displays a dialog box reading “5”

RETURN
Jumps to the instruction number specified by the top element of the stack, removes the top element, and resumes execution at the new instruction. The same operator, usually written with the name RETURN, returns after a subroutine call made by IF or CALL by jumping to the return address that the IF or CALL left on the top of the stack.

ROLL
Rolls the second element of the stack back the number of places specified by the top operand, then removes the top operand.
EX: “1 2 3 4 5 2 ROLL” rolls 5 back 2 places, leaving 1 2 5 3 4 on the stack.

RTI
Returns from interrupt. First enables interrupts like INTON, then pops return address from top of stack and jumps there like RETURN.
EX: “RTI” removes return address from stack.

SETINT
Sets an interrupt procedure to execute when interrupt occurs and interrupts are enabled. The top of the stack must have the name of

the register corresponding to the interrupt (choices are COLLISION, WALL, DAMAGE, TOP, BOTTOM, LEFT, RIGHT, RADAR, RANGE, and SIGNAL). The second item on the stack must be the address of a routine that handles the interrupt.
EX: "panicsub RADAR' SETINT" leaves nothing on the stack.

SETPARAM
Sets a parameter controlling when interrupts occur. The top of the stack must have the name of the register corresponding to the interrupt. The second item on the stack is the value to which the parameter will be set. Note that COLLISION and WALL ignore their parameters. The remaining default parameters are 150 for DAMAGE, 20 for TOP, 280 for BOTTOM, 20 for LEFT, 280 for RIGHT, 600 for RADAR and RANGE, and 0 for SIGNAL. SETPARAM may also be used to set information for the PROBE and HISTORY registers.
EX: "100 RANGE' SETPARAM" leaves nothing on the stack.

SIN or SINE
Sine function. The top argument should be the hypotenuse and the second argument should be the angle. Sine removes the top two arguments and returns the hypotenuse times the sine of the angle, truncated to an integer value. The angle must be between 0 and 359 or the result is undefined. Also, note that while 0 degrees is pointing straight up on the turret, the sine of 0 degrees is 0 (e.g. sin uses the standard trig coordinate system).
EX: "30 100 SIN" leaves 50 (=100*sin(30) on the stack.

SND0-9
Plays a sound defined in the Recording Studio. SNDx plays sound #x defined in the Recording Studio. Takes zero chronons to execute. The various other SND commands are used to enliven a robot with sound effects and to waste ungodly amounts of disk space.
EX: "SND2" does nothing to the stack but plays sound #2.

STO or STORE (either token is allowed)
Stores the second number on the stack in the variable specified at the top of the stack, and removes both the number and variable reference. The operand on the top of the stack must be a quoted variable or an error is reported. Also see the section on variables below, as values cannot be stored in some variables, such as RANGE.
EX: "20 aim' store" stores 20 in the variable AIM and leaves nothing on the stack.

SQRT
Square root function. Removes the top argument and returns its square root, truncated to an integer. If the argument is less than zero, an error results and the robot self-destructs.
EX: "10 SQRT" leaves 3 on the stack.

SWAP
Swaps the top and second elements on the stack.
EX: "1 2 SWAP" leaves 1 at the top of the stack and 2 in the second position.

SYNC
Ceases execution of code until the end of chronon. The next instruction will be executed at the next chronon. Has no effect on the stack.

TAN or TANGENT
Like SIN and COS, but computes the tangent function. If the result is greater than 19999 it is clipped to 19999; if the result is less than -19999, it is clipped to -19999.

VSTORE
Store a element in a vector (1 dimensional array). The top element specifies the element of the vector (0 to 100) while the second element is the number to store. Both arguments are removed from the stack.
EX: "42 1 VSTORE" stores 42 in the first element of the vector.

VRECALL
Returns the nth element of the vector. The top of the stack holds the element to reference; it is removed and replaced by the contents of the vector. Uninitialized vector elements default to zero. If the element number is less than 0 or greater than 100, zero is also returned.
EX: "1 VRECALL" leaves 42 on the stack assuming the previous example was last executed.

XOR or EOR (either token is allowed)
Exclusive OR. Checks if the top or second item on the stack, but not both, are nonzero, then removes them. If so, it places a 1 on the stack; otherwise it pushes a 0.
EX: "1 1 XOR" leaves a 0 on the top of the stack.

V: Registers

Each robot has a number of registers or variables. They are initialized to their appropriate values, or 0 if none is appropriate, when the battle starts. This section lists each variable, its use, and whether it can be read or written. The registers are:

```
A-Z         AIM         BOTTOM      BULLET      CHANNEL
CHRONON     COLLISION   DAMAGE      DOPPLER     ENERGY
FIRE        FRIEND      HISTORY     HELLBORE    ID
KILLS       LEFT        LOOK        MINE        MISSILE
MOVEX          MOVEY       NUKE        PROBE       RADAR
RANDOM      RANGE         RIGHT       ROBOTS      SCAN
SHIELD      SIGNAL      SPEEDX      SPEEDY      STUNNER
TEAMMATES   TOP         WALL        X           Y
```

A-Z (except X and Y)
User-defined variables. They may be used for any temporary storage that the robot needs. They may be read or written.

AIM
Angle turret points. May be read or written. The angle is in degrees, oriented like a compass with 0 degrees pointing upward and 90 degrees pointing to the right. All bullets and missiles are fired in the direction that the turret is pointing.

BULLET
Fires a normal bullet if either normal or explosive bullets are enabled. This is primarily useful so that robots equipped with explosive bullets can fire at short range without engulfing themselves in the explosion. Returns 0 if read.

BOTTOM / BOT
This register is used exclusively for interrupts. It has no effect if written and returns 0 if read.

CHANNEL
The robot's broadcasting and receiving channel. May be read

CHRONON
Returns the number of chronons elapsed in the current battle. CHRONON may only be read.

COLLISION
May only be read. If another robot has collided with the current robot, the COLLISION variable returns 1; otherwise it returns 0. When a collision with another robot takes place, both robots take one point of damage each chronon until they separate. They may either separate by changing direction, or by blowing the rival to little pieces.

DAMAGE
Robot's current damage rating. May only be read. When the battle begins, the damage rating starts at the maximum value set at the Hardware Store. Damage caused by bullets, missiles, and TacNukes that is not absorbed by the robot's shields is removed from the damage rating. When it reaches 0, the robot is dead.

DOPPLER
This register is used to help tracking routines. It is used in conjunction with the RANGE command. If no target is in the robot's sights, (i.e. the RANGE register is 0), DOPPLER returns 0. Otherwise, it returns the speed of the target in a direction perpendicular to the aim. For instance, if the target is moving directly toward or away from the robot, DOPPLER will return 0. If the robot's sights are pointing directly up and the target is moving from left to right at speed 8, the DOPPLER register will contain 8. DOPPLER has no effect if written. (Note: the DOPPLER command existed in some earlier versions of RoboWar, but was somewhat buggy and had not been documented.)

ENERGY
Robot's current energy. May be read, but not written. ENERGY returns the amount of energy the robot currently has. If not used for other purposes, energy is restored at 2 points per chronon. However, if the energy ever drops below 0, the robot does not interpret any more instructions or perform any more actions until the energy exceeds 0 again. When the battle begins energy is set to the maximum energy value specified in the Hardware Store.

FIRE
Used to shoot bullets. Returns 0 if read, shoots bullet with energy investment equal to amount written. This energy investment is removed from the robot's energy supply. It may exceed the robot's current energy value (placing the robot at negative energy and immobilizing it), but may not exceed the robots energy maximum. Depending on the settings from the Hardware Store, bullets may be normal, rubber, or explosive. Explosive bullets explode like TacNukes in a 36 pixel radius when they hit their target. Whey they detonate (3 chronons after impact) they do damage of 2*energy investment to all robots in the blast radius. This is a larger, faster explosion than in versions of RoboWar before 3.0. Normal bullets do damage equal to the energy investment when they hit their targets. Rubber bullets only do half damage if they hit. Bullets move across the screen at a speed of 12 pixels per chronon, heading in the direction that the robot's turret pointed when the shot was fired.

FRIEND
Is the robot sensed in a collision on your team? FRIEND can only be read. If the collision register holds a 1, the FRIEND register holds a 1 if the robot collided with is on the same team. Otherwise, the FRIEND register holds a 0. This is useful to determine if a collision has taken place with another robot on your same team.

HISTORY
Maintains a history of results and observations between battles. Each robot has 50 history registers. The history register to read or write is selected with the setparam command (e.g. 2 HISTORY' SETPARAM). The history registers are:

1) Number of battles fought
2) Kills made in previous battle
3) Kills made in all battles
4) Survival points in previous battle
5) Survival points from all battles
6) 1 if last battle timed out
7) Teammates alive at end of last battle (excluding self)
8) Teammates alive at end of all battle
9) Damage at end of last battle (0 if dead)
10) Chronons elapsed at end of last battle
11) Chronons elapsed in all previous battles

12-30) Reserved for future RoboWar versions

31-50) User-defined.

When no battles have been fought yet, numbers default to zero. User-defined history registers may be read or written and are preserved between rounds so robots can learn from the results of previous rounds. All other history registers may only be read. Suggestions for

new history registers are welcome. History is zeroed anytime robots are added or deleted in the arena (or between rounds in tournaments). It may also be erased or viewed with the history commands under the Arena menu.

Warning: in very long tournaments, it is conceivable that the history register could overflow 32767, the maximum integer in RoboWar. This is especially true for register 11. Hacker beware!

HELLBORE
Used to launch hellbores. Returns 0 if read, shoots hellbore with speed equal to amount written (this is different than hellbores in RoboWar 2.1.2). This amount is removed from the robot's energy supply. Hellbores reduce the shield of any robot they hit to zero but do no other damage. They must move at a speed from 4 to 20 in the direction that the robot's turret pointed when the shot was fired. Hellbores cannot be used unless they were first enabled at the hardware store.

ID
Robot's unique ID number. Each robot in the Arena has an ID from 0-5; it can be used to tell robots apart.

KILLS
The number of kills the robot has made in this battle. A robot gets no credit for killing itself or for crushing other robots during a collision.

LEFT
This register is used exclusively for interrupts. It has no effect if written and returns 0 if read.

LOOK
Targeting offset from AIM. The RANGE command returns a distance to the nearest robot in the direction AIM+LOOK. This might be useful for some tracking algorithm. If not otherwise set, LOOK defaults to 0. LOOK may be read or written.

MINE
Used to lay atomic land mines. Returns 0 if read, places a mine with energy investment equal to the amount written. The mine is stationary and becomes active ten chronons after placement. Once active, it will detonate against any target that hits it, causing damage equal to 2*(energy investment-5). This is twice as effective as mines in RoboWar 2.3 and earlier. Mines cannot be used unless they were first enabled at the hardware store.

MISSILE
Used to shoot missiles. Returns 0 if read, shoots missile with energy investment equal to amount written. This energy investment is removed from the robot's energy supply. It may not exceed the robot's energy max; if it does, only EnergyMax energy is used (this is changed; prior to RoboWar 4.1.2, a maximum of 50 points could be used). Missiles do 2*energy investment in damage if they hit their targets. Missiles move across the screen at a speed of 5 pixels per chronon, heading in the direction that the robot's turret pointed when the shot was fired. Missiles cannot be used unless they were first enabled at the hardware store.

MOVEX
Used to move the robot a given distance in the X direction without changing SPEEDX. Returns 0 if read, moves the robot the specified distance if written. Movement costs two points of energy per unit moved. Note that you can't fire a weapon of any type and move in the same chronon.

MOVEY
Used to move the robot a given distance in the Y direction without changing SPEEDY. MOVEY has the same characteristics as MOVEX. Note that you can't fire a weapon of any type and move in the same chronon.

NUKE
Used to place TacNukes, or Tactical Nuclear Devices. Returns 0 if read, places TacNuke with energy investment equal to amount written. This energy investment is removed form the robot's energy supply. It may exceed the robot's current energy value (placing the robot at negative energy and immobilizing it), but may not exceed the robots energy maximum. TacNukes begin to explode as soon as they are placed, increasing in radius by 5 pixels each chronon. At the tenth chronon, when they have a radius of 50, they detonate and cause 2*energy investment in damage to all robots in the radius. Robots who lay TacNukes are well advised to hasten away and be out of the blast radius when the devices explode. TacNukes cannot be used unless they were first enabled at the hardware store.

PROBE
Long range probe of opponent's systems. Returns information about the target in the direction of the AIM register when read; no effect if written. The register to probe is chosen with the SETPARAM command (e.g. SHIELD' PROBE' SETPARAM to select the SHIELD register for probing); it may be one of DAMAGE, ENERGY, SHIELD, ID, TEAMMATES, AIM, LOOK, SCAN. Probes must be enabled in the hardware store at the cost of one hardware point. PROBE defaults to probing the DAMAGE register if no other parameter has been set.

RADAR
Range to nearest bullet, missile, mine, stunner or TacNuke in the path of AIM. May only be read. RADAR checks a path 40 degrees wide centered on the AIM (actually AIM+SCAN, but SCAN defaults to 0). It returns the distance to the nearest bullet, missile, or TacNuke in this path. If there are none, it returns 0. Note that the weapon detected might be moving perpendicular to the aim, not toward the robot.

RANDOM
A random number from 0 to 359. May only be read.

RANGE
Range to nearest target in sights. May only be read. If there is a target in the direction the robot's AIM points, RANGE returns the distance. Actually checks in direction AIM+LOOK, but look defaults to 0. Otherwise, it returns 0.

RIGHT

This register is used exclusively for interrupts. It has no effect if written and returns 0 if read.

ROBOTS
Number of robots alive. Returns the number of robots alive in the arena, including the robot itself.

SCAN
Similar to LOOK, the radar offset from the AIM. The RADAR command searches for projectiles in the directon AIM+SCAN. This might be useful if a robot wants to look for targets in one direction, but check for danger in another direction. If not otherwise set, SCAN defaults to 0. SCAN may be read or written.

SHIELD
Robot's current shield level. May be read or written. If read, it returns the current level of the shield, or 0 if no shields are up. If written, is sets the shield level to the value written. If the current level is less than the level written, a point of energy is used for each point added to the shields. If not enough energy is available to set the shields, the shields are only strengthened as far as remaining energy permits. If the current level is greater than the level written, a point of energy is regained for each point of difference, although energy cannot exceed the maximum energy value set in the Hardware Store. Shields can absorb damage from bullets, missiles, or TacNukes that otherwise would have been deducted from a robot's damage score. Each point of damage that is done deducts one point from the shield level, until no power is left in the shields. The remaining damage is then done to a robot's damage score. Even if shields are not hit, they decrease by one half point each chronon from natural energy decay. Note that this replaces the old drain of one point per chronon in previous versions of RoboWar. Shields may be charged above the maximum shield value set in the Hardware Store (although they may never exceed 150), but if they are above maximum, they decrease by two points instead of one half per chronon. Shields are set to 0 when the battle begins.

SIGNAL
The signal value on the robot's current channel. May be read or written. If it is read, it returns the last value broadcast over the channel by any robot on the same team. If it is written, the value written is broadcast over the channel and may be read any time in the future by any other robot on the same team. Typically signals and channels are used by two or more robots to coordinate movement or team up against another set of robots.

SPEEDX
Speed of robot in left-right direction. May be read or written. Positive speeds move right, while negative speeds move to the left of the screen. If SPEEDX is read, it returns the current velocity; if it is written, it sets the velocity. Speeds must be in the range of -20 to 20. Each point of change in speed costs 2 points of energy; thus going from 10 to -2 costs 24 energy.

SPEEDY
Speed of robot in up-down direction. May be read or written. Positive values move down, while negative values move up. SPEEDY has the same limits and characteristics as SPEEDX.

STUNNER
Used to fire a stasis capsule. Returns 0 if read, shoots stasis capsule with speed 14 in the directon that the robot's turret points. The amound written is removed from the robot's energy supply; if a stasis capsule hits a robot, the robot is placed in stasis for one chronon for every four points of energy invested in the capsule. While in stasis, a robot does not move, interpret instructions, or regain energy; however, the robot's shields do not decay. Stunners cannot be used unless they are first enabled in the Hardware Store.

TEAMMATES
Number of living teammates, not including self. May only be read.

TOP
This register is used exclusively for interrupts. It has no effect if written and returns 0 if read.

WALL
Is the robot touching the electrified walls? Returns 1 when read if the robot is touching the wall, or 0 otherwise. No effect if written.

X
X position of robot. May range from 0 to 300 (the boundaries of the board). 0 is the left side; 300 is the right. X may be read but may not be written (no unrestricted teleporting!).

Y
Y position of robot. May range from 0 to 300. 0 is the top; 300 is the bottom. Y may be read but not written.

VI: Interrupts

Section III briefly described the value of interrupts for writing more efficient code. This section teaches how to actually write an interrupt-driven robot. Remember that interrupts are an advanced feature and are not necessary for a beginner to master before trying to write decent robots.

There are three crucial instructions needed by any robot that uses interrupts: SETINT, INTON, and RTI. SETINT tells the processor what label to jump to when the interrupt occurs. INTON turns interrupts on. By default, interrupts are off when a robot first begins combat, so the INTON instruction must appear in the robot's code before interrupts may occur. RTI (ReTurn from Interrupt) is used in place of the RETURN instruction at the end of interrupt-handling routines. RTI reenables interrupts before returning.

Let us begin with a simple example robot, Bozo.

```
# Bozo
# Written 8/22/93 by David Harris
# This robot uses interrupts.
```

```
    killem RANGE' SETINT
    INTON

main:
    AIM 5 + AIM' STORE
    main JUMP

killem:
    50 FIRE' STORE
    RTI
```

The first line in the program sets the RANGE interrupt to killem. This causes the robot to automatically call the killem routine when it sights a target (if interrupts are enabled). A complete list of available interrupts appears later in this section. INTON turns on interrupts. The main loop just makes the turret spin.

When a poor target comes into Bozo's sights, i.e. when the RANGE register is non-zero, the RANGE interrupt is triggered. This causes the robot to leave a return address on the stack (just like an IF or CALL statement happened), disable interrupts, and jump to the killem label. The robot will fire a shot with 50 energy. The RTI statement returns from the interrupt by first reenabling interrupts, then jumps back to the return address stored on the top of the stack. RTI is equivalent to an INTON statement followed by a RETURN statement, but is more convenient.

There are three other instructions related to interrupts. One is INTOFF. It disables interrupts until the next INTON instruction. This might be useful in time-critical operations where a robot cannot afford to inadvertently take an interrupt. Another is FLUSHINT, which dumps all pending interrupts from the queue (see below for more information on the interrupt queue). This might be useful if the robot is about to change its course of action and doesn't want the baggage of stale old pending interrupts. The third is SETPARAM. It sets a parameter describing when a particular interrupt should occur. For example, suppose we inserted the following statement right after SETINT:

```
100 RANGE' SETPARAM
```

This sets the parameter on the RANGE interrupt to 100, indicating that a RANGE interrupt should only occur when the RANGE register is less than 100. The various parameters that may be set are also described below. Each interrupt has a default value for its parameter that can be changed if desired by SETPARAM.

If an interrupt is no longer needed, it can be turned off without disabling all other interrupts by setting it to -1, as in the statement:

```
-1 RANGE' SETINT
```

All interrupts are initially set to -1 by default.

Summary of Interrupts

The following interrupts are supported in order of highest priority to lowest:

COLLISION
Sets the collision interrupt, to occur whenever the collision register of a robot changes from 0 to 1. SETPARAM has no effect on the collision interrupt.

WALL
Much like collision, but occurs when a robot runs into a wall. SETPARAM also has no effect.

DAMAGE
The damage interrupt is triggered whenever a robot takes damage. SETPARAM sets the minimum threshold required for the damage interrupt to occur; by default it is set to 150. This is useful if a robot should only change its behavior when it is damaged beyond a certain point.

SHIELD
The shield interrupt is triggered whenever a robot's shield transitions from above to below a predetermined threshold. SETPARAM sets this threshold; by default it is 25.

TOP
The top interrupt is triggered whenever a robot moves too close to the top wall. SETPARAM determines the y coordinate at which the interrupt is triggered; by default, it is 20. Note that the directional interrupts are only triggered once when the robot crosses the threshold.

BOTTOM (or BOT)
The bottom interrupt is triggered whenever a robot moves too close to the bottom wall. SETPARAM determines the y coordinate at which the interrupt is triggered; by default, it is 280.

LEFT
The left interrupt is triggered whenever a robot moves too close to the left wall. SETPARAM determines the x coordinate at which the interrupt is triggered; by default, it is 20.

RIGHT
The right interrupt is triggered whenever a robot moves too close to the right wall. SETPARAM determines the x coordinate at which the

interrupt is triggered; by default, it is 280.

RADAR
The radar interrupt is triggered at the beginning of a chronon or when the aim or scan registers change and the RADAR register is nonzero. SETPARAM determines the maximum distance at which a projectile will set off the interrupt; by default it is 600 to trigger on anything.

RANGE
The range interrupt is much like RADAR, but triggers at either the beginning of a chronon or when the aim or look registers change and RANGE is nonzero. SETPARAM determines the maximum range that will trigger an interrupt and is also 600 by default.

TEAMMATES
The TEAMMATES interrupt is triggered whenever the robot's teammate is killed. The robot can specify for the interrupt to only occur when the number (excluding yourself) is below a particular value by using SETPARAM. For instance, to only interrupt when all of your teammates are dead, set the parameter to 1. By default, the parameter is set to 5, causing an interrupt when any teammate dies.

ROBOTS
The ROBOTS interrupt is triggered whenever a robot is killed. The robot can specify for the interrupt to only occur when the number (including yourself) is below a particular value by using SETPARAM. For instance, to only interrupt when all other robots are dead, set the parameter to 2. By default, the parameter is set to 6, causing an interrupt when any robot dies.

SIGNAL
The signal interrupt is triggered when data is broadcast over the communication channels by a robot's teammate. SETPARAM determines which channel number is being checked; by default it is channel 0. It is generally a good idea for different robots to transmit on different channels to prevent one robot from overwriting the data sent by the other. Also, since multiple messages that are rapidly sent might be lost, it is generally wise for the RoboTalk hacker to devise some protocol for teammates to acknowledge each other's transmissions.

CHRONON
This interrupt is triggered at the start of each chronon, starting at the chronon set by the parameter. By default, the parameter is set to 0. This interrupt is useful for animated icons or to change behavior after a specific amount of time.

Interrupt Priorities & the Interrupt Queue

Since several different interrupts may occur at the same time, it is important to understand the system RoboWar uses to manage multiple interrupts. The two key concepts are the priority levels of interrupts and the interrupt queue.

Each interrupt has a particular priority. From highest priority to lowest, the interrupts are: COLLISION, WALL, DAMAGE, SHIELD, TOP, BOTTOM, LEFT, RIGHT, RADAR, RANGE, TEAMMATES, ROBOTS, SIGNAL, CHRONON. If two interrupts occur at exactly the same time, the one with higher priority is processed first.

The interrupt queue is a list of interrupts waiting to be processed. If interrupts are disabled (for example, while one interrupt is being handled) and a new interrupt occurs, it may be placed in the queue to be processed when interrupts are reenabled. If there are several interrupts pending in the queue, the one with highest priority is handled first when interrupts become reenabled. Note that RANGE, RADAR, and CHRONON interrupts are never queued (and hence, are not affected by FLUSHINT). Also note that at most one interrupt of each type is stored in the queue; e.g. if a robot is damaged twice while interrupts are disabled, only one damage interrupt will occur when interrupts are reeanbled. Finally, if an interrupt procedure is set to -1 (the default beginning state, or reset explicitly by SETINT), interrupts of that type will not be placed in the queue.

Most interrupts are added to the queue only at the beginnings of chronons. RANGE and RADAR are the exceptions; they can occur midchronon when a robot moves its turret, and are never added to the queue.

VII: Tournaments

About twice a year, an International RoboWar Tournament is held. Hackers of all ages from around the world submit their best creations to duel for the coveted championship. The rules tend to change slightly from tournament to tournament, so be sure to check your mail about the precise rules. This section describes the tournament utility and general tournament information. It also contains the RoboWar Hall of Fame, listing RoboMasters who have won past tournaments.

Tournament Utility

The new, improved tournament facilities introduced in RoboWar 4.1 support several types of tournaments: complete tournaments, team tournaments, basic tournaments, and test tournaments. The clunky old custom tournaments have been eliminated; if there is something else you really want the tournament utility to do, send in a suggestion. Complete tournaments are similar to what has been called "individual tournaments" in the Fourth through Ninth RoboWar Tournaments (May 1992-January 1995). Team tournaments are also the same as they have been for years. Basic tournaments are like complete tournaments but don't have the Winner's Circle. Test tournaments are for your convenience testing your latest robot against a group of oppontents. Scoring can be done the same as it has been in the past, but there is a new option for "aggressive scoring" which rewards more violent robots.

To run a tournament, choose Tournament from the Arena menu. Choose the type of tournament and enter a filename in which to store the results, then click OK. Select a set of robots from the standard file dialog or choose a custom tournament file. The tournament will run automatically and results will be written to disk.

Each type of tournament except Team tournaments is divided into duels and group rounds. In duels, robots fight one-on-one. In group rounds, randomly selected groups of six robots slug it out. Standard scoring rules give one point to each robot that is alive at the end of the duel (after 1500 chronons, the battle is declared a double win), 3 points to any robot alive at the end of a group battle, 2 if the

robot outlives 4 other robots, and 1 if the robot outlives 3 other robots. A tie gives the robot the higher score. Hence, robots can do very well by simply surviving, rather than attacking other pacifists.

The scoring system can be summarized as follows:
Kill Points: 1 Point is given to a robot for every other robot that it kills provided it live for 20 chronon's after the kill.
Survival Points - Individual: 1 Points is given to each bot alive at the end of the last chronon.
Survival Points - Group: 3 Points is given to each robot who is alive at the end of the battle. If a robot outlives 4 Robots, but is then killed it gets 2 survival points. If a robot outlives 3 robots and is then killed it receives 1 survival point.
Team: Teams are scored as individual battles where each Robot on a team gets survival points and kill points. The team's score is the sum of both robot's points.

While this scoring system reflects many aspects of the real world, it can lead to boring tournaments. To encourage more bloodthirsty robots, the "aggressive scoring" system gives an additional point for each kill they make, provided they live for 20 chronon's after the kill. Suicide and crushing the opponent during a collision do not count as kills. Kills of your teammates do not score your robot points.

RoboWar 4.1.2 adds an "official rules" check box. Under official tournament rules, undocumented features are prohibited and will cause your robot to crash. You can turn this off if you really want in battles with friends, but entries to the official International RoboWar Tournaments must work under official rules.

Team tournaments have similar scoring. In the standard team rules, each team gets one point if at least one member is still alive at the end of the battle. In the aggressive rules, a team gets one point for each member living at the end of the battle plus one more point for each kill a teammember makes.

The results are printed in a plain text file that you can view with your favorite text editor or word processor. Duels and group rounds can be repeated multiple times to reduce the role of chance in the results. A Tournament Log file is also produced for detailed analysis of tournaments (except team tournaments). The log file lists complete results of every battle. Advanced RoboWarriors may write scripts that process the logs and extract information of interest, such as win percentages against various opponents.

Complete Tournaments

Select a folder full of robots for the complete tournament by choosing one robot in the folder. The complete tournament consists of 4 stages. First, each robot fights every other robot in a duel. Then a bunch of group battles with randomly selected participants are run. The group scores are normalized to compensate for some robots fighting in more groups than others and to scale the average points for a robot in group battle to approximately equal the average from the duels.

The six robots with the highest scores are selected for the Winners' Circle. These six robots fight each other in duels and as a group of 6 and the scores are again normalized to average half of the average from the first stage duels. The robot with the highest sum of points from all four stages is declared the winner.

Complete tournaments will be used to run the Individual category of future International RoboWar Tournaments.

Team Tournaments

Select pairs of robots to participate in the team tournament, then choose cancel when all teams are selected. In the team tournament each team fights every other team and scores are summed to determine the winner. In the future, support may be added for groups of 3 teams to fight, but that feature is not currently implemented.

Basic Tournaments

While designing a robot, you may wish to check how it performs without going through the complete tournament and waiting for winners' circle results. This tournament lets you select the number of times to run a set of robots in duels and group battles. Group scores are normalized as in the complete tournaments. You can also run your robot against five others to simulate a winners' circle. Click on a robot in a folder to select all of the folder's robots for the tournament.

Test Tournaments

For fastest testing, you may wish to test one robot against many others. Select one robot to pit it against each other robot in the folder in duels and against randomly selected opponents in group combat. The tournament will print out your performance against each opponent, your percentage of wins in duels, and group scores of each robot normalized by the number of groups in which it fought.

Tournament Procedure

A typical tournament has three categories: individual robots, teams of two robots, and icons. Individual robots battle with the complete tournament setup. Teams of two battle with the team tournament command. Icon judging is generally done by a squad of montly volunteers munching while watching the other rounds and can be highly subjective. The icon contest title is generally awarded for the best combination of animated icon and sound; of course, entering copyrighted sounds or icons with a robot is illegal without special permission. I reserve the right to delete large sounds from robots before distributing them with the other tournament entries if disk space is tight.

Some tournaments also features the youth category, giving special recognition to the best individual entry written by a hacker under 18 years old. Little League and Titan contests, for bots with 2 and 21 hardware points respectively, were held in the Fifth and Sixth tournaments, but will not be held in the future unless interest picks up sharply.

Many people guard their secret RoboTalk techniques by deleting their source code before distributing their robots. While this is fair enough, one may claim even more fame and glory by distributing a prize-winning robot with complete code. Thus, novice hackers can learn from your example and you can put all sorts of funny messages through your code. Of course, if you borrow code from

somebody's robot, you should properly attribute it. A letter to the author might be polite and wholesale copying of the bulk of a robot is obvious unethical.

RoboWar Hall of Fame

The RoboMasters, winners of the past tournaments, are named below:

Tourney	Category	Author	Robot
I	Ind 1st	Matt Sakai	Silo Plus
I	Ind 2nd	Dave Gangon	ShazBot
I	Ind 3rd	Tim Seufert	Timbot V Mark II

Tourney	Category	Author	Robot
II	Ind 1st	Jon Newman	Rammer AMT
II	Ind 2nd	Tom Morrison	Chicken & Corn
II	Ind 3rd	Doug Harris	MX™ III
II	Team	Doug Harris	Mx3
		Tim Seufert	Mortician®
II	Icon	Wesley Voshell	Patriot

Tourney	Category	Author	Robot
III	Ind 1st	Robert Hogg	Charger IV
III	Ind 2nd	Robert Hogg	Whumper
III	Ind 3rd	Robert Hogg	Blazer
III	Team	John Baylis	-Knuckles-
			Sword&Shield
III	Icon	Tim Seufert	Calvinism II

Tourney	Category	Author	Robot
IV	Ind 1st	Doug Harris	Pacifist Penquin II
IV	Ind 2nd	Robert Hogg	Beholder Jr.
IV	Ind 3rd t	Eric Foley	Orb of Doom
IV	Ind 3rd t	Robert Hogg	Lewis
IV	Team	Tim Seufert	Terminator II
		Doug Harris	Miracle Penquin
IV	Icon	Robert Hogg	Lewis

Tourney	Category	Author	Robot
V	Ind 1st	J. Rommereide	Excelsior
V	Ind 2nd	Robert Hogg	Lug
V	Ind 3rd	Eric Foley	Orb of Doom II
V	Team	Robert Hogg	$*Lucky Flea*$
			$*Nuclear Flea*$
V	Titan	Eric Foley	Krulockh Lord
V	L. League	Doug Harris	wimp
V	Icon	Robert Hogg	Dirty Lewis

Tourney	Category	Author	Robot
VI	Ind 1st	A. Duchateau	Sylvestre
VI	Ind 2nd	S. Linhart	◊Darling◊
VI	Ind 3rd	Eric Foley	Dragon Knight
VI	Team	S. Linhart	◊Darling #1◊
			◊Darling #2◊
VI	Titan	Doug Harris	z Death Penquin
VI	L. League	S. Linhart	◊Lil Darling◊
VI	Icon	Eric Foley	Eye of Shorshirsh

Tourney	Category	Author	Robot
VII	Ind 1st	Paul Schmidt	Vortex v2.2
VII	Ind 2nd t	Paul Schmidt	Cloak Folk's Revenge
VII	Ind 2nd t	Andrew Lindsey	His All Seeing Eye III
VII	Team	Paul Schmidt	Cloak Folk's Revenge
			Team - WonderDog

Tourney	Category	Author	Robot
VIII	Ind 1st	J. Lechat	Delsevart
VIII	Ind 2nd	J. Lechat	Jade
VIII	Ind 3rd	Rob Williams	The Artful Dodger
VIII	Team	Doug Harris	Chlorinated Benzene Man
			Chlorinated Benzene Woman
VIII	Youth	John Abbott	Superzot II
VIII	Icon	Mark Ramirez	Brave Sir Robin

Tourney	Category	Author	Robot
IX	Ind 1st	J. Lechat	Jade
IX	Ind 2nd	Jeff Lewis	Horta Jr.

IX	Ind 3rd	J. Lechat	Arachnée
IX	Team	John Abbott	.Team Superzot III
			.Team Superzot Jr
IX	Youth	John Abbott	.Superzot Jr.
IX	Icon	Chris Funston	Immortal

Tourney	Category	Author	Robot
X	Ind 1st	J. Lechat	Jade
X	Ind 2nd	Doug Harris	*CL79*
X	Ind 3rd	J. Lechat	Carne
X	Team	Colin Jaffe	Killer DogCow #1/#2
X	Youth	Greg Parker	HappyBot™ XXI nuke
X	Icon	Parker Bros.	PokeyMorph Team

Tourney	Category	Author	Robot
XI	Ind 1st	Colin Jaffe	SPAMBot 7 oz.
XI	Ind 2nd	Andrew Clinton	Defense Drone
XI	Ind 3rd	J. Lechat	Jade
XI	Youth	Colin Jaffe	SPAMBot 7 oz.
XI	Titan	Colin Jaffe	SPAMBot 12 oz.
XI	Team	Colin Jaffe	Occam's Razor
			Sword of Damocles
XI	Icon	Paul Mount	Çat5

Tourney	Category	Author	Robot
XII	Ind 1st	Denys Seguret	Dialectix
XII	Ind 2nd	Andrew Clinton	S'pht'kr
XII	Ind 3rd	Paul Olson	Merantes
XII	Titan	Eric Foley	Nightshade
XII	Team 1st	John Abbott	Plecostomus #1/#2
XII	Team 2nd	Alex Landratis	MOLDer & SKULLy
XII	Icon	Nathan Paymer	Zany Zygote

Tourney	Category	Author	Robot
XIII	Ind 1st	Seth Zenz	Magic Sword
XIII	1st tie	Eric Foley	Obsidian
XIII	Ind 3rd	Denys Seguret	Dialectix
XIII	Titan 1st	Eric Foley	B-ko
XIII	Titan 2nd	Eric Foley	Nightshade
XIII	Team	Eric Foley	Obsidian twins
XIII	Icon	Taro Yamamoto	HaruBot

Tourney	Category	Author	Robot
XIV	Ind 1st:	Paul Hansen	Automaton
XIV	Ind 2nd:	Noah Hoffman-Smith/	
		Paul Olson	The Porcelain God
XIV	Ind 3rd:	Thong Nguyen	Vigor
XIV	Titan 1st:	Paul Hansen	Plis'dkU
XIV	Titan 2nd:	Eric Foley	B-ko
XIV	Team 1st:	Eric Foley	Death Bunnies III
XIV	Team 2nd:	Adam Locke-Norton	Mayo Twins
XIV	Icon:	Patrick Moor	Tchernobyl

Tourney	Category	Author	Robot
XV	Ind 1st:	Alex Landraitis	ItWasLate&IWasTiredBot
XV	Ind 2nd:	Christian Thalmann	Invincibiliti
XV	Ind 3rd:	Alex Landraitis	DaveHarrisSavesTheDay
XV	Titan:	Christian Thalmann	LanceBot
XV	Teams:	none	
XV	Icon:	Alex Landraitis	DaveHarrisSavesTheDay

Tourney	Category	Author	Robot
XVI	Ind 1st:	Alex Landraitis	Half-CensoredUpgrageBot
XVI	Ind 2nd:	Adam Locke-Norton	Anarchy
XVI	Ind 3rd:	John Abbott, Jr.	30 Minutes
XVI	Titan 1st:	Eric Foley	Mar
XVI	Titan 2nd:	Adam Locke-Norton	Mr. Melon
XVI	Team:	Adam Locke-Norton	Dave #1 and Dave #2
XVI	Icon:	Mark Wagner	Sitting Duck

The hackers with the greatest number of titles are:

1. Eric Foley - 13
2. Robert Hogg - 9
3. Doug Harris - 8 (two shared with Tim Seufert)

```
4. Jean-Francis Lechat - 7
5. (tie> Colin Jaffe - 5
5. (tie) Alex Landraitis - 5
5. (tie) John Abbot, Jr. - 5
8. (tie) Tim Seufert - 4 (two shared with Doug Harris)
9. (tie) Paul Schmidt - 3
9. (tie) Stephen Linhart - 3

Or, to be more specific . . .

Greatest Mortal-RoboMaster Of All Time:
  Jean-Francis Lechat (7 prizes)
Greatest Titan-RoboMaster Of All Time:
  Eric Foley (6 prizes)
Greatest Team-RoboMaster Of All Time:
  Doug Harris (3 prizes) (two shared with Tim Seufert)
Greatest Icon-RoboMaster Of All Time:
  Robert Hogg (2 prizes)
Most Dominant Combatant, One Tournament:
Two with four apiece -
  Colin Jaffe (Tournament XI) and
  Eric Foley (Tournament XIII)
Most Dominant Combatant, Consecutive Tournaments:
  Robert Hogg (9 prizes from Tournaments III to V)
Most Tournaments With Titles Won, Consecutive:
  Jean-Francis Lechat (4 - VIII to XI)
Most Tournaments With Titles Won, Total:
  Eric Foley (7 - IV, V, VI, XII, XIII, XIV, and XVI)
Most Complete Sweep Of A Tournament Round:
  Eric Foley (top 7 places, titan round of Tournament XII)

For more information about the tournaments and RoboWar visit the RoboWar Hindquaters at: http://
www.teleport.com/~stiltman/RoboWar
```

VIII: A Brief History of RoboWar

Many moons ago, the world was lacking RoboWar. Then one crisp spring night a bunch of fanatical teen-age computer programmers were sitting around at a meeting of the Ridgecrest IEEE Student club tossing out programming ideas and dodging the shrapnel. Sick of hand-eye arcade games, the concept of robots fighting without user intervention grabbed their attention. Thus was born RoboWar.

The computer club members debated many ideas. At first they considered placing robots on a board full of obstacles and challenges. They thought about building robots by dragging icons about to form various subsystems. Jon Richards was the first to prototype RoboWar code on a foul MS-DOS clone. Later that summer at Caltech, David Harris saw an early version of another robot-battle style game being developed by students there. He liked the concept of the programming language for robots and saw a number of other ideas for improvement.

In the fall, while climbing Dragon Peak out of Onion Valley in the Sierra Nevada, David was hit across his head with a fascinating, efficient, and easy to implement programming language based on Reverse-Polish notation. Fortunately, another computer club veteran, Ralph Giles, was also on the hike and the two worked out the details of the language, with inspiration from the HP-28 calculator language, C, Pascal, and assembly language in addition to their own foul concoctions. Fortunately, they wrote a complete specification of the interpreter, for, although Ralph spent a bit of time working on it on another clone, it wasn't until December that David began to implement it on a real machine.

David first designed an interpreter and compiler that ran, albeit clumsily, on a Silicon Graphics workstation. Then he ported it to the Macintosh, built up a user interface, and spend most of Christmas vacation chasing bugs. When the program worked reliably, progress greatly slowed, for he spent extensive time "playtesting" instead of coding.

Since that point, robots have evolved through a number of stages. This history details the development of code by the IEEE club members. Doug Harris, David's demented brother, also built a number of robots, ranging from unreliable to excellent in quality, that followed some convergent and some divergent branches of evolution. The IEEE club robots have come through six generations:

First generation robots include MoveBot (listed below) and DumBot. They were written to test the interpreter on the Silicon Graphics. They duel nicely but fairly randomly. Each was generally less than a page in length and wasn't smart enough to kill its opponent in a single shot. TimBot (also listed below), a simple but effective robot, lead the second generation of robots into existence. Written by Tim Seufert on the Macintosh, it locked onto stationary robots and blew them to tiny bits. Originally, it used less energy in its shots and took longer to eliminate its opponents, but still, TimBot made completely stationary robots unviable.

In response to TimBot, drooling hackers started to produce a third generation of robots. These typically moved about the board. One interesting pair of third-generation robots were Coroner 1 and 2. These 'bots moved to opposite corners before shooting at any targets they saw. If they began taking damage through their shields, they would flee to the opposite corners.

Matt Sakai changed the entire course of robot evolution when he brought Silo IV to a meeting on a nondescript floppy disk. Silo IV, the one and only fourth generation robot, moved about the board and almost unfailingly destroyed its first, second, and third generation opponents. For a time, the IEEE group was stunned; David even added some more features to the interpreter that robot designers might try because Silo IV seemed to be the ultimate robot in the evolutionary tree.

Nonetheless, after a few weeks, numerous other robots using similar tactics sprung from the silicon mind. These fifth generation robots, the so-called "Silo clones", eventually managed to equal, then best Matt's Silo IV. Among the Silo clones were Robot B2 (the product of David Wasserman's nightmares), TimBot IV (at first the joke of the IEEE group, later a reasonably effective robot), Freud (most famous for his remarkable icon), and Blade (the best Silo clone yet developed). To test these robots, the IEEE members ran extensive combats and developed the "TimBot Test:" how many plain-vanilla TimBots on one team can one robot defeat at one time in at least 60% of the combats? Most Silo clones could almost always defeat a single TimBot and could defeat two more than 60% of the time.

Fortunately, before things became too dull, robots using other strategies began to appear. Among these sixth generation robots were Lewis Girod, Aeneas II, and Pearl. Aeneas broke the three TimBot barrier. Unfortunately for his creator, Pearl appeared soon after, usually capable of overcoming five other TimBots teamed together in a battle! Pearl, another product of Matt Sakai, is the current champion.

What is the future of RoboWar? Pearl appears to be very good but Silo at one point also seemed unbeatable. Few robots have been written to work effectively together on teams; few use the communications or advanced projectile detection capabilities. Perhaps somebody will develop tracking algorithms or discover another excellent algorithm lurking just out of sight. Th to explore. For the novice, RoboWar has much excitement waiting as one learns to program and to overcome each generation. For the expert, too, the horizons of RoboWar are still beckoning.

In such a state RoboWar rests April 16, 1990. I will be sponsoring a RoboWar tournament in early June in which the best robots of each contestant may compete for glory and riches. See the about box for more details. Also, please pay your shareware fees if you enjoy RoboWar. I put a great deal of time and effort into programming. Since the $10 fee is so much less than the cost of a professionally marketed game, I would appreciate the money.

History Continued. On June 1st, (tonight, as I write) we are collecting the robots for the tournament. We received 11 entries at $2 each. Matt introduced his reputedly fearsome seventh generation "Religion" robots-Hinduism, Judaism, and (The Great God) Anything, replete with wild comments as well as powerful new designs. Yes, there are robots better than sixth generation Pearl clones! We are waiting for the robots from Northwestern University (hopefully Jon will send some) and plan to hold the competition very soon. YAWP!

Continued: The RoboWar competition is complete! Twelve robots entered; their names, owners, and comments are listed below:

Robot	Creator	Comment
ShazBot	Dave Gangon	The Physicist
Line II	Wesley Voshell	De Best
TimBot V Mark II	Tim Seufert	Kills Strafers
Hinduism	Matt Sakai	Strafer that'll kill TimBot anyway
Judaism	Matt Sakai	No comment.
Anything	Matt Sakai	Convert or Die!!!
Silo Plus	Matt Sakai	Like the icon?
Blade	Ralph Giles	The old battle-ship
Schön	Ralph Giles	How much potential is realized?
Line	Matt Mann	----------
Appletree	David Wasserman	Cannon fodder
Tower Bot	Doug Harris	Have a nice day

Jon never sent any robots (finals were the week before the robots were due) but Doug entered his Tower Bot which had a nasty habit of overflowing the stack just after the 3500 chronon battle limit was over and thus forcing the battle rosters to be redone. Fortunately, with the help of Ralph's Mac IIci, a version of RoboWar using the 68881 instructions, and the automated combat features, the tournament only took a morning to complete.

First Place goes to Silo Plus, Matt's seventh-generation shapechanger. ShazBot took second place with a good algorithm to escape wall huggers and to track Silo clones. Finally, the shield-bearing Silo clone TimBot V Mark II surprised us all and took third place.

From this contest, we find the old guns still do fairly well. Blade, the fifth generation robot, far outperformed his cousin Schön, the sixth-generation perimeter 'bot. Matt had a good design in his seventh-generation robots, though. They were typically very efficient wall-huggers that changed behavior (thus the name shapechanger) as the game progressed.

Over the next several months, RoboWar received only sporadic work. The 500 instruction limit was clearly a limiting factor for new robot development, so raising the limit to 5000 was a crucial part of the new RoboWar 2.0 program. Also, the Central Control metaphor, though intuitively simple, required a large number of mouse clicks to navigate and violated the standard Macintosh interface pretty badly. The biggest change to RoboWar 2.0 was a reorganization of the user interface to be (hopefully) much faster to use.

David also began receiving shareware checks and letters about RoboWar. Letters from new RoboWar addicts asking for new features were very exciting and spurred on development of RoboWar 2.0. The program was revised by the end of January but these instructions needed a rewrite. Unfortunately, with the semester starting, David didn't have time for the full rewrite that the instructions deserved, but hopefully this document is accurate and clear enough to suffice for the present.

What is the future of RoboWar? RoboWar 2.0 is so full of new features that I'm afraid some will turn out to be Intel Features (a.k.a. bugs). Over the next few months, I'm planning to collect bug reports and post fixes on the network (at sumex-aim.stanford.edu for all you Unix types out there). Davitime permits.

And now for the results of the Second RoboWar tournament. This tournament's entries were quite exciting, stretching RoboWar 1.5.1 to its very limits. Eighth generation robots who specialized in evading enemy shots were common in this contest. The contest had sixteen individual entries plus David's old standards, Freud and Aeneas III, placed in the arena for comparative purposes. Rammer AMT, by Jon Newman, scored a clear first place victory. Chicken & Corn II, by Tom Morrison, overcame his moderate performance in the initial rounds to place second. Finally MX™ III, by Doug Harris, placed third with his TacNuke superbombardment. Mx3 and Mortician®, a team jointly

written by Doug Harris and Tim Seufert, clearly won the team competition. Patriot, by Wesley Voshell, won the Icon Competition with a marvelously drawn picture of a ship. Congratulations to everyone who entered the contest. This competition featured some of the best robots I had ever seen. With the new flexibility of RoboWar 2.0, I expect to see even more exciting robot designs.

In the spring of 1991 I ran a class on RoboWar at MIT for junior high and high school students. Each week the students would program robots and bring them to class to compete with their fellows. I observed how the robots evolved over time; each week there would be minor improvements over the previous week's robots but every once in a while, somebody would introduce a significantly different robot technology (a "paradigm shift") that could defeat most all previous robots. The new robot would spur another series of incremental improvement until finally there was another paradigm shift. I also found that the chance for students to interact and observe each other's robots helped the students to develop much better robots. By the end of the class they had programmed robots that could compete well with the entries to the second tournament. RoboWar proved to be a valuable educational tool for teaching programming techniques to fairly young students. I would like to thank the members of my class for all the bugs they helped me track down and for the insight they gave me on RoboWar learning patterns.

In July, Dave “Rasferet” Blumenthal, a Cornell student, entered the RoboWar Hall of Fame by cracking the RoboWar password system. As promised, Dave is now due for infinite glory and undying mention in the sacred RoboWar Instructions. Dave has also made numerous suggestions and bug reports that have helped my development of RoboWar very much.

In May 1992, the Fourth RoboWar Tournament was held, featuring some extremely well-designed robots. Doug Harris, whose clever designs in the past were fatally wounded by bugs, won both the individual (Pacifist Penquin), and (with the help of a robot by Tim Seufert) team (Miracle Penquin & Terminator II) prizes. Robert Hogg again demonstrated his RoboWar mastery taking second place with Beholder Jr., tieing for third with Biosphere, and winning the icon contest with the utterly cute Lewis. Eric Foley's Orb of Doom also tied for third in the individual contest. This time there were a number of effective teams; moreover, many of the robots had extremely good icons and not a single entry performed very badly. Jesse Barnum discovered a remarkable new way to cheat, but the bug was fixed before the tournement so he just receives an honorable mention.

Two weeks after the tournament, misplaced mail surfaced at the East Campus desk. Enclosed was Jeff Rommereide's Excelsior, his Fourth RoboWar tournament entry. As the results had already been mailed out, I was unable to entere Excelsior in the tournament, but I staged a mock tourney anyway. Excelsior claimed first place! This was quite tragic; Jeff
had invested a great deal of effort in new technology for this robot and it performed exceptionally well. Thus, I promised Jeff a glowing description of Excelsior in this manual and expect that it will be entered in the next RoboWar tournament to challenge hackers. Congratulations, Jeff: may Excelsior claim eternal glory in the halls of RoboWar Heroes!

A few weeks later, the impossible happened: I finally got around to implementing the RoboTalk debugger! YAWP!

A large contingent of RoboWar hackers negotiated a site license at Carleton College and ran their own local tournament in October 1992. Tom Hayes sent me some of the results; Nereid was the winner and Leech, an annoying 98 instruction bot, took second. Little VT 002 won the Little League division. I do not have any of the robots or any more information, but you can enquire with Tom at THAYES%ADMIN1@carleton.edu. If any other groups are interested in RoboWar site licenses or run noteworthy local tournaments, drop me a letter...

The Fifth Tournament was somewhat smaller, but featured a number of excellent robots. The individual winners were (from first to third): Excelsior by Jeff Rommereide, Lug by Robert Hogg, and Orb of Doom II by Eric Foley. Robert also won the team contest with $*Lucky Flea*$ and $*Nuclear Flea*$; these and other suicidal destroyer teams dominated the contest. In the interest of promoting strategy over luck, suicidal destroyers will be banned from future tournaments. Eric won the first titan competition with the dreaded Krulockh Lord and Doug Harris was winner of the little leagues with Wimp. Finally, Dirty Lewis tickled the funny bone of the judges and claimed supremacy in the icon contest!

After a long hiatus from RoboWar programming (taking a heavy load at MIT does wonders for gobbling up all free time), I finally got back to adding my favorite features from people's wish lists during finals week of Spring 1993. These features appear in RoboWar 2.4. They include the Recording Studio for adding sound to robots, interrupts (which may revolutionize robot design), and lots of fixes of small but annoying bugs reported over the last year. Thanks to everyone who contributed suggestions or bug reports; your ideas help make RoboWar a better program.

The Sixth Tournament, held June 1993, claimed the distinction of being the first International Tournament. Three of the twelve people entering the contest came from overseas and many of their robots did very well. The Darling series of profoundly aggressive robots ushers in a new generation of robot designs that has never been seen before. The individual winners were (from first to third): Sylvestre by Antoine Duchateau, Darling, by Stephen Linhart, and Dragon Knight by RoboMaster Eric Foley. Stephen Linhart also won the team and Little League categories with a pair of Darlings and with Lil Darling, respectively. RoboMaster Douglas Harris won the Titan competition with the latest of his Penquin series, z Death Penquin. Finally, Eric Foley won the icon contest with his all-seeing and unpronouncible Eye of Shorshirsh.

RoboWar never was released as version 2.4; instead, I took the leap up to version 3.0 by adding automatic math coprocessor support, interrupts, and so forth. Version 3.0 should be released in September 1993.

RoboWar 3.0 introduced almost as many new bugs as new features. Thanks go to the many RoboWar hackers who sent detailed bug reports that have helped me squash these problems. Some names particularly worthy of recognition are Stephen Linhart, David Pokorny, and Tim "the enchanter" Mattox. Keep those bug reports coming!

Version 3.1, released at the end of January, 1994, fixes many of these bugs. Unfortunately, the dreaded Thesis prevented me from adding any of a long list of desired new features, but one of these days, Version 3.2 should come out with any remaining bug fixes and with a couple of minor new features.

At this point, the artistry of Brian and Greg Parker should be mentioned in the RoboWar Hall of Fame. These two intrepid young RoboWar hackers drew nifty scenes of robots blasting each other to oblivion on the envelopes in which they sent various pieces of correspondence. Nice drawing!

Andrew Lindsey's "His All Seeing Eye III" (entered in the 7th tournament) takes advantage of an undocumented feature and a glitch in RoboWar to pull a really neat trick. Try tracing its code in the debugger and watch what happens!

In May 1994, I finally upgraded from trusty old Mac II to a spiffy new PowerMac 8100. Using the excellent Metrowerks compiler, I ported RoboWar to run native on the PowerMac. Video isn't much faster, but when the battle isn't displayed, it runs up to 30 times faster than it did on my old machine! Nice job, Apple!

The Eighth International RoboWar Tournament, held in July 1994, was the largest and best so far. Interrupt bots have come of age and many clever strategies are employed.

RoboWar 4.0 evolved over the summer of 1994. The Metrowerks compiler is an excellent product and the programmers obviously are craftsmen who love their work. Unfortunately, it isn't as thoroughly tested as other compilers and has produced bad code on occasion. Metrowerks is very good about fixing bugs that are reported, but I am afraid that sneaky bugs might be left in RoboWar that I haven't caught. Thanks go to Tim Seufert and Ben Matasar for beta testing RoboWar 4.0 and catching lots of bugs!

One of the most interesting (or perhaps least interesting, if you have a slow Mac) results of the Eighth International Tournament was the evolution of cooperation among robots. The scoring system gives one point to any robot that survives in a duel. RoboWarriors have gradually realized that killing the opponent doesn't help one's own score; hence a number of robots have emerged that don't attack unless they sense aggression from the other party. Eric Foley describes these robots as the "Alliance." Such alliance members can often outscore even the best aggressors simply by scoring lots of points in each round they encounter fellow alliance members. This situation is a variant of the Prisoner's Dilemma and also has many parallels in the real world, where peaceful nations can advance economically while aggressive ones mortgage their future for new bombs. It is fascinating to see such real world cooperation emerging from a simple simulation like RoboWar where no communication is possible save for observing the opponent's actions. Unfortunately, the Alliance leads to many long boring rounds where the robots simply watch each other for 1500 chronons before a draw is declared. To appeal to our lust for senseless virtual violence, the scoring system may change in the future to reward effective killers as well as robots that live to a ripe old age.

RoboWar 4.1 features new registers and operators and an improved tournament utility. Jeff Lewis of Bakersfield, CA deserves credit for pointing out lots of bugs in the old tournament routines and suggesting the nifty new tournament interface. The history registers evolved while contemplating extended games of Prisoners' Dilemma while hiking Mission Peak. They should open up lots of creative strategies, but risk increasing the complexity of a world-class robot. Thanks to Eric Foley's urgings and lots of boring Alliance members in the Ninth Tournament, a new aggressive scoring system has been instituted that rewards robots for kills as well as survival.

RoboWar 4.1.1 and 4.1.2 mostly fix bugs. In September, 1995, I moved to Stanford to begin my Ph.D. RoboWar business has picked up in 1995 (largely thanks to the growth of the net) and my mother Sally Harris has generously offered to handle registrations so I'll have more time to design microprocessors. The 4.1.2 release reflects the new RoboWar HindQuarters address (back at the RoboWar birthplace in Ridgecrest, CA) and a new email address that will hopefully be processed more rapidly than my Stanford one.

The Tenth International RoboWar Tournament continued to be dominated by Jean-Francis Lechat, the Napoleon of RoboWar. Jade held out in the lead and Carne demonstrated a very novel use of drones. For fairness, drones and lasers will no longer be allowed in future tournaments (because Doug Harris complained they violate the "no undocumented features" rule). Who will devise a robot clever enough to overcome Jean-Francis' minions in the next tournaments?

The Tenth Tournament also had a loophole in the rules that allowed self-destructive robots in the Individual competition to rack up many points in group combat. To patch this loophole and encourage survival in general, RoboWar 4.1.2 only awards points to robots for kills if the aggressor is still alive at the time of the kill. Also, some robots used lasers and drones. These weapons are now in the category of "undocumented features" and are banned from future official tournaments. Finding them is a challenge for the hacker; please do not ask me about how to enable or use them or to bring them back into the game as an official feature.

RoboWar 4.1.2 was finally released in December 1995 after I tracked down and squashed an annoying bug that caused projectiles to move at different speeds depending on where they were in the arena. The bug was difficult to find because it only happened on certain Macintoshes and because some of the reports about the bug seemed to have incorrect information. Moreover, the bug did not occur on my development machine, so I had to find a machine where it did happen and port my development tools. In the end, the bug turned out happen on 68K Macs with floating point hardware; it was due to letting the floating point routines get out of date with respect to the normal routines. It should be fixed now. RoboWar 4.1.2 also fixes a systematic bias that caused robots early in the alphabet to do slightly better in tournaments.

Thanks to RoboMasters Eric Foley, Jean-Francis Lechat, and Greg Parker, as well as Lucas Dixon and numerous other RoboWar hackers, for detailed bug reports that greatly stabilized RoboWar 4.1.2.

Eric Foley has developed a very nice web page with a taxonomy of RoboWar strategies. His categories are much more up to date than this history. It can be reached from the RoboWar web page.

The Eleventh Tournament chronicled the rise of Dashers, robots which capitalize on the aggressive scoring rule. In particular, the SpamBot series of dashers built by Colin Jaffe dominated the tournament. Many RoboWarriors became discouraged, fearing that the Twelfth Tournament would feature only incremental improvements in dasher technology and that new strategies were unlikely to arise. Denys Seguret surprised the RoboWar world with his innovative Dialectix robot which soundly crushed the dashers. Meanwhile, Eric Foley swept the titan round with his horde strategy of robots which recognize their friends. Hordes were novel once, but will be discouraged in future tournaments because they essentially amount to buying the victory with a large entry fee. The Thirteenth Tournament should be exciting as dashers and anti-dashers and new strategies vie for the title.

RoboWar 4.2 was released in January 1997, offering battle replay (for easier testing) and various minor improvements and fixes. Thanks to many who suggested improvements and pointed out bugs, including Colin Jaffe, Thong Nguyen, Lucas Dixon, Andrew Clinton, Paul Olson, Eric Foley, Steven Merrill, Amn Arah J. Leonard (who gave the instructions a much-needed proffreading), Bryant Brandon, Tom Livak, and Denys Seguret. At this point I am trying to stabilize RoboWar. Many people have sent in suggestions about interesting

weapons and modifications to the arena, but I am not currently considering such changes because they will upset the balance currently existing in the arena.

After at least four years of promising color icons and being too lazy to deal with the finer points of Color QuickDraw, I took up Lucas Dixon's offer to implement them in spring 1997. He did a pretty slick job with it, leading to RoboWar 4.3. The difficulties of managing two versions of the code lead to some bugs (hopefully we've tested 4.3 sufficiently, but I never know until I turn the bugs loose on the world), so I'm not too eager to farm out development in the future, but Lucas Dixon and the beta testers deserve thanks for a much-needed job. Incidentally, Lucas added a default icon of a warrior for new robots. The icon is from Realmz, a fantasy game from Fantasoft. Thanks to Fantasoft for permission to use the icon; check out their web site at http://www.fantasoft.com. RoboWar 4.3 also fixes a bug involving negative energy which was exploited in an unofficial tournament. Remember that exploiting bugs in official tournaments is illegal and will result in disqualification (and the Golden BozoBot Award if you try to hide it).

Eric Foley hosted the Thirteenth International RoboWar tournament. There were fewer entries than usual, but the quality was very high. Eric swept a bunch of the top positions (no, he didn't cheat!) elevating himself to holding the all-time largest number of RoboWar awards. He's also developed a thorough history of RoboWar tournaments which can be reached from the RoboWar Web page.

Unfortunately, three more individuals receive the Golden BozoBot Award. After being clearly told that exploiting the negative energy bug of RoboWar 4.2.1 would be illegal in the Thirteenth Tournament, Nathan Paymer did so anyway and tried to conceal his action. Ever-vigilant Foley caught the cheating and nominated the robots for the BozoBot award. More seriously, Matt Wellstein's Twelfth Tournament entries (Eliminator, Shimmiebot, and Razorback) plagerized code extensively from Cloak Folk's Revenge, Darling, and Vortex of previous tournaments. Matt's comments suggested that he had invented the code himself, but in actuality it was copied almost completely. I sent mail to Matt requesting that he apologize to the author, but got no reply. Matt receives one of the worst Golden BozoBot Awards in the history of the game for stealing code, claiming it as his own, and declining to apologize. Thanks to the alert RoboWarrior who pointed out the theft. Similarly, David Eisenstat stole the exact code from Jade and entered it in the Eleventh Tournament under the name Dave, forever disgracing the noble name.

RoboWar 4.4 is a transition, which fills me with mixed feelings. I've been maintaining and improving RoboWar for eight years now. It's been very exciting to meet so many RoboWarriors around the globe. It's also been profitable enough to pay for my development computers and software, though shareware is never a way to get rich. I've watched RoboWar evolve as I've moved from high school to MIT to professional engineering and back to graduate school. It's a perfect lesson in large-scale software engineering and in the challenges of maintaining code over eight years of changes. My biggest regret is hard-coding lots of calls to the Macintosh OS into the code, which made RoboWar run fairly fast on the Mac Plus but also means RoboWar is nearly impossible to port to other platforms. I've grown too lax with upgrading RoboWar, though, so for the sake of the game I've decided it is time to pass it on to new blood, namely Lucas Dixon. Lucas is a bright and enthusiastic hacker from Scotland who's agreed to continue improving the game for some time to come. I'm sure you'll read his musings in future entries to this history.

As my first entry into the RoboWar history I'd like to thank David Harris for doing such a great job. His name will remain a legend in RoboWar for all time.

After a couple of minor fixes producing RoboWar 4.4.2, I decided to take on the Drafting Board in a complete rewrite to use the WASTE text engine. This with a few other changes and a preferences dialog box, and a Major, at the time, and much disputed, change to the Game Engine to disallow robots from being able to move and shoot in the same chronon made RoboWar 4.5. Which was more buggy than I had hoped for, and had some unexpected conflicts with versions 7.5.1 to 7.5.3 of the MacOS. Tournament 16 braught new life to RoboWar having stopped the game killing dasher paradim.

Started in late 1998 a complete rewrite of RoboWar in C++ is making it possible to port RoboWar to Windows (urk!) and Linux (Yay!). This is expected to have it's first release in the summer of 1999. I'm also hoping that it will make RoboWar more stable, easier to change in the future and will also let new languages be developped. I'm particularly interested in Prolog, a language much-used by the University of Edinburgh's Artificial Inteligence department where I am studying.

Remember that RoboWar thrives because of you, the clever RoboWarriors who continually seek to overcome one another with smarter, stronger, and more efficient robots! Keep up the innovation that makes RoboWar so unique among computer games!

Appendix A: Version History

Robots are not the only beasts which evolve with time. The RoboWar application has gone through numerous versions, expanding features, patching bugs, and introducing new and wonderful bugs. This version history records the changes from version 1.5 onward.

Version 4.5.2 (May 9, 1999)

- Random number generater is now properly random.
- Debugger icon no longer draws to the frontmost window even when in the background.
- rti allows jumps to negative positions in code - fixed, it now gives an error as it should.
- Find: can cause a infinite loop = crash. and can't find anything after 32 - fixed.
- Fixed a Compiler bug that sometimes tries to compile a robot of more than 5000 instructions (should bring up an error), can cause a crash.
- Fixed occational cosmetic bug in chronon per second display.
- Normal is now limmited to 60 chonons per second.
- pagedown/pageup doesn't work in Drafting board.- Fixed. You can now use pageup and pagedown.
- File type outputted by robowar for tourney logs and tourney info is no longer of the wrong type.
- Cosmetic Bug in prefs dialog fixed.
- Fixed bug where you still weren't allowed to move after shooting with the moveshoot limmitation turned off. Rules should now do what they say. :)
- RoboWar *should* now work on MacOS7.5.1 to 7.5.3, but no been able to test. :(

Version 4.5.1 (Dec 15, 1998)

- Print disabled. It didn't work. Will get working again in next version.

- RoboWar crashed when told to goto an error in a bugy bot on 68k machines, fixed now.
- Score display works properly - how'd I miss this one?
- RoboWarPPC still doesn't work on system 7.5.1ppc, you have to use the 68k version for some still unknown reason.
- Rules menu added under Edit. It lets you change the rules RoboWar uses (move shoot, Lazers, Drones).

- Shift-Click now works in the drafting board.
- Tournament log type still doesn't do anything, will be fixed for next version.
- Other small or not so small bug fixes.

Version 4.5 (Aug 5, 1998)
This is a bug update, whith many changes, and hopefully not too many bugs...

- Icon and sound commands no longer count limmit mac code length, though there is a limmit of no more than 100 icona nd snd instructions per robot.
- Auto shield icon option in the icon factory.
- Game Engine Change: can't move and shoot in the same chronon. This was implemented to stop dashers... we'll see how well it works.
- Preferences dialog. This will be added to in future, and currently holds some basic game preferenecs.
- lazers and drones back in the compiler, but are unofficial, bugs with them are also fixed.

- sin/cos/tan noiw accurate to a signle degree.
- Scoring Alterations.
  - Syntax Colouring: not dynamic, but still useful, config from prefs.
  - Improved Battle points display, now it's coloured and displays this battles points and total points
  - IconFactory colour selection is bigger.
  - 32k limmit on the size of a robot's text has been removed.
  - Help moved to help menu.

Version 4.4.2 (January 17, 1998)
Fixes minor bugs adds one new feature.

- Scoring system change: points are aworded after a robots death unless they overloaded.
- Fixes minor messge bug caused by 4.4.1
- Added Battle Points to Robot Display.

Version 4.4.1 (December 15, 1997)
This is the first version to be released by Lucas Dixon. It is purly a bug fix release.

- Fixed bug to do with jumping out of the robot's code, stopped robot's from being able to jump negative code, in interupts, jumps, if's, calls.
- Fixed mine bug which gave a mine negative energy when no negative energy and mine power zero was used.
- Improved interface for color selection in the icon factory (selected color is shown...) and option-click on the color selction control uses the system color selector.
- fixed color selection bug, on monitors with more than 256 colors.
- added option-click for eye dropper shortcut and control click for fill shortcut.
- Fixed 101 drone bugs... namely: they can now hit robots, do damage, they don't 'jiggle' and they have the correct icons. fixed spinning drones bug (drones spinned when they hit someone).
- ColorSelection window will now stay on the screen when it is placed close to the edge.

Version 4.4 (October 16, 1997)
This is the last version produced by David Harris. Lucas Dixon will be in charge of future releases. The changes in this version are mostly for the convenience of programmers:

- Count highlighted instructions whenever compiling
- Produce tournament log for further analysis
- Explicit RECALL instruction available
- Optional icons available for collision, death, icon hit, shield hit
- Fixed bug causing shots to miss target
- Removed "Sound purged" dialog
- Preserve the tournament name you entered

Also, a RoboWar mailing list is now available. To join, send the message "subscribe robowar" to majordomo@lists.stanford.edu.

Version 4.3.1 (June 8, 1997)
Fixes a memory error in 4.3 which causes crashes on some machines and fixes a glitch in robot explosions.

Version 4.3 (May 27, 1997)
This version adds long-promised color icons, thanks to Lucas Dixon who implemented them in England! It also fixes a bug in RoboWar 4.2.1 Arena which was exploited by Adam Locke-Norton to win the Pre-Dawn Skirmish Tournament.

Version 4.2.1 (February 4, 1997)
This version fixes a serious bug in the tournament utility which replays battles.

Version 4.2 (January 26, 1997)
This version adds a replay feature for battles, plus a number of smaller changes and fixes:
Features:

- Holding option key replays last battle
- Removed MOVEX and MOVEY distance limits
- TACNUKEs and explosive bullets do 2x damage
- PRINT statements time out during tournaments
- Tournament final group round runs slightly faster
- Arena displays killer of dead robots
- Directives to turn off warnings about STORE with unquoted variable

Bug fixes:

- Missles over 127 energy now do proper damage
- Interrupts cleared properly on robot death
- High color depths handled with less memory
- Icon Factory right shift corrected

See the new section in the RoboTalk part of help for information on the assembler directives.

Also, effective with this release, notification of tournaments and new versions will only be sent by email (exceptions may be made if you live someplace where email is unavailable and have specifically requested to get snailmail notice). When your email changes, send a note to RoboWarHQ@aol.com to keep us informed of your new address.

Version 4.1.2 (September 23, 1995)
This release contains many small bug fixes and improvements including:

- Tournament fixes: minimum number of entries, fixed scoring bug, banned undocumented features in official tournaments
- Mines: mines arm 10 chronons after being laid
- Save as: eliminated faulty warning
- Debugger: eliminated negative energy bug
- Missiles: may have up to EnergyMax of energy assigned
- Probes: initialized to measure damage at beginning of battle
- Scoring: no points for kills after you die or of your teammates
- Arena: randomized order of robot evaluation for fairness, fixed bug that causes projectile speed to vary
- Printing: fixed bug in printing instructions
- Vectors: VSTORE and VRECALL access elements 0-100, not just 1-100

Thanks to all the RoboWar hackers who have reported these bugs.

Version 4.1.1 (April 30, 1995)
This minor release fixes an uninitialized variable that sometimes caused tournaments to hang. It also improves printing by leaving margins at the edge of the paper and fixes a minor bug in the debugger.

Version 4.1 (March, 1995)
This version adds many new registers and operators and sports a much-improved tournament utility (again!) The HISTORY registers may revolutionize the design of top-ranking robots.

- New FLUSHINT, MAX, MIN, ARCCOS, and ARCSIN operators
- New PROBE, HISTORY, ID, and KILLS registers
- Improved tournament handling
- Fixed debugger to begin at start of battle
- Missiles do 2x damage (honest, they do now)
- RoboWar Web Page documented
- Minor functional fixes
- Minor cosmetic improvements

Version 4.0 (Fall, 1994)
Native PowerMac support is the major new feature of this RoboWar release. With all video off, RoboWar can execute up to 750 chronons / second native on my PowerMac 8100! Unfortunately, my new compiler cannot selectively generate code for the 68881, so older Macs with math coprocessors may notice a slowdown. `:-(` These nicely formatted instructions are also immediately evident. A complete list of other changes follows:

- Missiles do 2x damage
- Added TEAMMATES register
- Added SHIELDS, TEAMMATES, ROBOTS, CHRONON interrupts
- AppleEvents support added; later versions should be fully scriptable
- Alphabetized registers and operators
- Added instructions on tournaments, registration, Hall of Fame
- Better chronons / second display for fast battles
- Keyboard shortcuts for showing battle and playing sounds
- Touranment automatically sorts results
- Boosted registration fee to $15
- Minor bug fixes including ARCTAN, DOPPLER, arrow keys, Recording Studio, dialog boxes, cosmetic arena changes

Known bugs:

- Sounds are still flakey-turn off sounds in the Arena menu if they crash your system

Version 3.1 (January 31, 1994)
This version fixes several bugs found in version 3.0 involving stunners, sounds crashing Centris machines, radar returning the wrong value at long ranges, radar and range appearing incorrectly in the debugger, robots with processor speed 30 being unopenable, and inconsistent instructions.

Sound is still known to be a problem; sometimes sound turns on and off of its o

Version 3.0 (August 22, 1993)
The additions to version 2.4, combined with new interrupt routines that may revolutionize RoboTalk programming, have been promoted to Version 3.0 of RoboWar. Several new instructions and registers have been added and several wimpy existing weapons have been souped

up to be more competitive. The precise changes are:

- Interrupt Support
- New Instructions: inton, intoff, rti, setint, setparam, dropall
- New Registers: wall, top, bot, left, right
- Fixed & Documented Register: doppler
- Explosive bullets expand to size 36 in three chronons
- Mines do twice as much damage as before
- Zoom to buggy line in Drafting Board when a robot crashes
- Minor bug fix involving collision handling
- New undocumented feature

Note that the format of the Hardware Store resources has changed, so although RoboWar 3.0 can read the robots of previous versions, RoboWar 2.3 may incorrectly read the hardware resources of robots from version 3.0. Also, the Recording Studio produces occasional error messages of the nature "Error Disposing Sound." These shouldn't be happening, but can generally be ignored. If trouble persists, check "Don't Play Sounds" under the Arena menu.

Version 2.4 Beta (July 25, 1993)
Many old bugs have been fixed in this version; moreover, many new features have been added that will probably be accompanied by their own bugs. The most important changes are: the Recording Studio for adding digitized sounds to robots, the new Stunner weapon, and a general speedup of the program (time-critical routines have been improved for about a 25% speedup and 68881 support has been added for roughly a 50% speedup on Macs with floating-point hardware). A complete list of changes follows:

- Many small bug fixes in the RoboTalk interpreter
- Reduced flicker in the Debugger
- Battles can run in the background
- Icon editor supports flips and rotations
- Recording studio supports recording and copying sounds
- Processor Speed 30 option added
- Save As... works correctly
- Lasers and Drones no longer available (too pathetic)
- Stunners available
- Arena runs faster and supports floating point hardware

Version 2.3 (June 7, 1992)
This version introduces the new RoboTalk debugger, something promised for a year and a half now! The DEBUG instruction is also available. This manual has been somewhat updated to define Little League and Titan class tournaments and a few other minor changes. If one sets that maximum advantage value under the Arena menu to 99, cheating robots are permitted; i.e. ResEdit hacks to modify hardware values are allowed. Errors in printing an

Version 2.2.1 (May 15, 1992)
The Fourth RoboWar tournament brough to light several subtle bugs in version 2.2 that are fixed here. This version fixes a bug that causes battles to end prematurely and also corrects a glitch that allows robots to fire bullets of negative energy that do huge amounts of damage!

Version 2.2 (March 24, 1992)
The tournament utility is now complete. It currently supports both old custom tournaments and the new automated forms for the Fourth RoboWar Tournament. Error alerts now time out after 30 seconds so a crashed robot won't tie up an unattended tournament.

Version 2.2 Beta (January 24, 1992)
This is a preliminary version of the next major release of RoboWar. Hellbores now behave differently; this change is not backward compatible. Robots now have 50 processor speed rather than 30. One can grab (click on) a robot during a battle and move it about for debugging purposes. Shields are more efficient, decaying at only 1/2 point per chronon. A bug in Save As has been corrected so robots save properly to other folders, and a few other small arena bugs have been corrected. In order to compile a robot of more than 100 lines, one must register for RoboWar and obtain a code number. The maximum allowable hardware points may now be user-specified, creating the possibility of "Titain" (unlimited advantages) or "Little League" (very few points) competitions. The instructions can be saved to disk by the user. Finally, the initial hooks for a source level debugger are installed. Still missing, however, is the debugger and the improved tournament utility.

Version 2.1.3 (July 8, 1991)
Ye olde bugs strike again. This version corrects a problem with radar detecting targets located straight above a robot and fixes another error in the collision routine that sometimes prevents collisions from registering. For the fourth or fifth time, I hope that the collision detection bugs are thoroughly crushed! This version also addes color icons for System 7 and adds a cancel box to the print dialog box so that users can cancel out-of-control print loops without resorting to Ye Olde Programmers' Switch. When leaving one of the editors, RoboWar now only updates the last modification dates if changes have actually been made.
The IFG, IFEG instructions and the DOPPLER register are now available.

Version 2.1.2 (April 17, 1991)
This version makes a number of small fixes and improvements. It repairs a serious bug that prevents robots from having more than 100 labels. It lets battles start with a single robot for testing purposes. Finally, it displays the chronon at which a given robot died.

Version 2.1.1 (April 7, 1991)
This version fixes a bug that causes RoboWar to crash while playing sounds on older Macs. It fixes the sine and cosine operators so that they return the actual sine or cosine of the angle and fixes a bug in the roll operator. Also, it adds the DIST operator to compute distances.

Version 2.1 (March 31, 1991)
This version adds sound to the Arena. It also adds a new instruction (BULLET) which lets robots with explosive bullets also fire regular bullets. It checks that arithmetic operations stay within the bounds of legal numbers. Finally, it adds these instructions on line accessible from the help command.

Version 2.0.1 (March 23, 1991)
This version fixes some severe bugs discovered in version 2.0. It fixes printing, which caused a bomb on 68000 machines, and catches a memory allocation error which could trash robots accidentally. 2.0.1 also makes some minor changes to the update routines to reduce unnecessary flashes and glitches.

Version 2.0 (February 3, 1991)
A major redesign of RoboWar, attempting to keep maximal compatibility while greatly increasing the range of possible robot designs and fixing some long-standing bugs. A synopsis of the new features appears below; more information can be found in appropriate sections of the manual above. WARNING: Robots edited in the RoboWar 2.0 hardware store are no longer readable by version 1.5.1 (due to a change in the format of the hardware information). Also, robots that go below -200 energy melt their power supplies and blow up. This fixes a loop hole in 1.5.1 but means that some robots, like MX™ III and Lich Sr. no longer work. These two changes are the only incompatibilities I know of between 1.5.1 and 2.0.

New Instructions: LOOK, SCAN, ABS, CHRONON, SIN, COS, TAN, SQRT, ICON0-ICON9, PRINT, SYNC, ROBOTS, FRIEND, VSTORE, VRECALL, HELLBORE, DRONE, MINE, LASER. Also, the maximum number of instructions is now 5000 and the maximum number of labels is 400.

Bug Fixes: Collision detection bug squashed (on the nth attempt), bullets no longer "jump over" the edge of their targets, the almost-invisible yellow robot is no longer a color used in the Arena.

Battle changes: Robots graphically explode, battles don't end until 20 chronons after the robot dies (to allow retaliatory shots to hit), robots with bugs or robots that cheat blow up instead of halt the combat, a chronons per second count monitors the speed of the battle.

User interface: The user interface has vastly changed. Check out the new about box, too...

Version 1.5.1 (May 28, 1990)
This version is a bug fix on version 1.5. Most significantly, I found that the DUP command does not work in the old versions. This causes great problems and confusion for robots using DUP. Now DUP behaves correctly. I also fixed some glitches in the password protection, regarding opening robots. I also added tallying features for group battles to the automated combat menu. Finally, I updated these instructions to include some of these advanced features.

Version 1.5 (May 18, 1990)
This version introduces two major new features: automated combat and password protection. In anticipation of the upcoming tournament, I added these features to help run the large number of combats and to keep everyone's code secret. Notice: I hate passwords, so if anyone has their robot garbled by the password protection, I'll just laugh. Version 1.5 also adds the "Don't show battle" menu option. This speeds up battles significantly, especially on a color monitor. After extensive testing I found that 256 color mode slowed RoboWar down the greatest amount during most combats; however, a large number of bullets on the screen at one time (as used by Matt's Pearl and his seventh generation robots) slows the game even further. Worse yet, not showing the battle scarcely improves the performance of missiles. I dread running large numbers of combats with his robots. They usually take at least ten minutes. A final change: a number of intrepid RoboWar hackers discovered a method of cheating to create nearly invincible robots by going through a back door in the program. Unfortunately for them, I now check for this method of cheating and tweak any robots who violate the rules. Don't worry: you won't stumble upon this method of cheating by accident.

Appendix B: Sample Robots

The following code segments are the programs for the four sample robots distributed with this disk, Stationary, DumBot, TimBot, and MoveBot:

```
# Robot Stationary
# 12/30/89 By David Harris

main:
    aim 10 + aim' store
    main jump
```

••••••••••••••••••••••••••••••••••

```
{
    DumBot
    Created 12/27/89 by David Harris

    This robot maintains shields, rotates its
    turret, and fires when it finds a target.
}

Main:
    50 shield' store
    aim 7 + aim' store
    range 0 >
       missilesub if
    main jump
```

```
MissileSub:
    50 missile' store
    return
```

••••••••••••••••••••••••••••••••••

```
#  Tim's Robot
#
#   Designed by the same person
#   Who, at the present, due to the presence
#   of altogether too many Electric Monks,
#   believes that he is a banana and that Dodo
#   is more powerful than Mac II, thereby
#   causing his programming ability to
#   deteriorate.  Oh well.

    random aim' store

Main:
    range 0 = rotate shoot ife
    main jump

rotate:
    aim 17 + aim' store
    return

shoot:
    energy 20 > reallyshoot if
    return

reallyshoot:
    energy fire' store
    return
```

••••••••••••••••••••••••••••••••••

```
{ MoveBot
  Created 11/21/89 by David Harris.

    This robot moves about the screen,
    maintaining shields and searching for a
    target.  It fires when it sights anything.
}

START:
     1 speedx' store
     1 speedy' store
     25 shield' store

MAIN:
     aim 5 + aim' store          # Rotate Turret
     x 50 < xmin if                   # X minimum
     y 50 < ymin if                   # Y minimum
     x 250 > xmax if                  # X maximum
     y 250 > ymax if                  # Y maximum
     range 0 > shoot if          # Shoot if range >0
     25 shield' store
     main jump

XMIN:
     random 3 mod 1 + speedx' store
     return

YMIN:
      random 3 mod 1 + speedy' store
     return

XMAX:
     -1 random 3 mod - speedx' store
     return

YMAX:
     -1 random 3 mod - speedy' store
     return
```

```
SHOOT:
     energy 2 / missile' store
     return
```