Contents

What is it?

(Begin: 1 of 2)

This is a multi-player strategy game combined with a genetic algorithm engine. Any combination of human and computer players can play a game. Computer players can be individually programmed or collectively evolved using genetic algorithms. The computer players are evolved not during a game, but over the course of playing many thousands of games with each other. The evolution engine keeps a pool of computer player genes (programs), which it pits against each other in four-game tournaments. It rewards the more successful genes by allowing them to combine as parents of new programs which replace the less successful genes. With this game, your computer opponents can literally improve on your computer while you run it.

PLAYING THE GAME

Use your armies to conquer the board. Some board areas have factories, and factories are needed to support your armies. The more factories you capture, the more armies you can build--to capture more factories. You cannot see where your opponents' forces are unless you encounter them with your armies, and once encountered, opponent forces limit the movement of your armies. You can build spies to wander freely through enemy lines and see how the other players have distributed their forces. Spies can often show you a weak area to break through. But beware, your enemy can see your spies, so all watchers are being watched!

What Should I Do First?

(Begin: 2 of 2)

THE VERY FIRST THING

If you've never played, or you've only watched someone else play, it would be a good idea to familiarize yourself with the sections under The Game Rules and The Game Play Interface, you don't have to read every word, just quickly browse the sections to see what is there.

PLAY A GAME OR TWO

Then, start a new game playing Red, with just one human (you) in the game. Choose the OK button on the GAME SETUP screen, and play a few games to get the feel of it. During the game you can bring up The Game Rules and The Game Play Interface sections again, and this time as you try things, or don't know how to do something, you can read the details.

TRY THE AI ADVISOR

While you are playing, see what the RED.DNA program would do by pressing **F7** or selecting **Orders|Ask the AI** from the menu. If you don't like what it does, you can use your mouse to change only the orders you don't like, or select **Orders|Cancel all orders** to revert back to no orders. You might build a spy or two and move them around behind enemy lines to see what the other players are doing. Sometimes an opponent who attacks viciously does not protect all areas on his front lines. Find a weakness with one of your spies, and go get some of his factories!

CHANGE THE AI ADVISOR

Get into a game a few turns, and then select **View|Gene Editor**. In the Gene Editor window, select **Run!** from the menu bar. This does exactly the same thing as selecting **Orders|Ask the AI** from the main game window, but now you can change a number or two in the program, and when you select **Run!** you can immediately see what your change did to your orders, if any. You do not need to understand the Gene programming language to do this. To learn about the Gene programming language, start with the Concepts section under Writing Computer Player Programs in this help file.

WATCH WHAT THE COMPUTER PLAYERS DO

Set up a game with 4 human players, making sure to select advisors for each, or use the defaults. Start the game. When you get to the select player dialog, press the **SPACE BAR** to get the next available player, the first time this will be RED. Now you can press **F7** to see what the RED advisor would do. From now on, you can press the sequence: **F9**, **SPACE BAR**, **F7**, to cycle through and see what each computer advisor would do. You can step through an entire game this way.

EVOLUTION 101

Once you are familiar with the game, read Evolution in Cloak, Dagger and DNA and then bring up the Evolution Lab Screen while you scan the other sections under Evolving Computer Players in this help file, or use the HELP buttons from the Evolution Lab Screen. If you haven't done anything since installing, press the FILL button on the Gene Pool tab page to generate an initial set of genes from the original four. Leave the original four locked, you'll see why later. Choose some parameter settings, at first you may want to leave them at the default values. Choose Continuous Play in the Run Mode box and Allow Evolution in the Engine box. Now press the RUN button and watch what happens!

AFTER A FEW THOUSAND TOURNAMENTS

When you've let it run long enough, the original four genes' food supply will begin to go negative. This means they are consistently losing games, and it also means you have **evolved better players!** You can continue this process, unlock the losers, to free up the space, and lock some of the ones who have played quite a few games and still have a high score. Then let it run some more. Of course you can skip the locking and unlocking and just let it run.

After a while, EXTRACT three of the best genes to DNA files, and then start and play a game, (with you as a human player of course) and compete against the other three! Are they harder to beat than the

originals?

There is an element of randomness in the evolution engine part of this program. Often long periods of time can go by with no apparent evolution. Then a short spurt of improvement(s) and then another static phase. This seems to be normal with Genetic Algorithm programs, and has been referred to as "punctuated equilibrium". The random number generator is initialized by the time on your computer when you run the program, so it is unlikely that you will have the same results as anyone else even if they do exactly the same thing. If any of your friends are running this program, trade your best DNA files once in a while!

ABOVE AND BEYOND THE SCOPE OF THIS PROGRAM

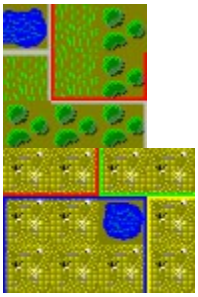
One of the problems with Genetic Algorithms is that it is not always obvious what is the best selection and reproduction criteria. In other words, what values should you put in the parameter settings page? Certainly there has been work done in this area, and perhaps research already done (in the proceedings listed in the Bibliography section of this help file?) would indicate a better approach than has been taken. The Evolution Lab as it stands now, reproduces by demerit rather than merit, meaning that new genes are created only when other ones run out of food, not directly because they were successful. Would evolution proceed more efficiently if the successful genes (by whatever criteria) were always allowed to reproduce, and genes were deleted only when space was needed?

One interesting idea with Genetic Algorithms is to use them to solve some of the above problems. Imagine that the parameter settings (and any other interesting controls/methods you can think of) are yet another genetic code. An arbitrary starting pool could be created, which would be evolved for some fixed number of tournaments, say fifty thousand. In the same way that the Evolution Lab calls the game engine to play entire games with the computer player programs as the genetic codes, some other program could call the Evolution Lab to run fifty thousand tournaments, with the parameter settings as the genetic codes, always with the same starting pool. You would need some method of evaluating the resulting pools for comparative fitness, just as the individual genes are compared by playing games. You could then run this program to "evolve" better parameter settings for the Evolution Lab.

The Board

(The Game Rules: 1 of 5)

TERRAIN AND BORDERS



The board consists of a map of bordered areas, where the areas contain a number of square terrain cells. Two areas are considered adjacent if there is a section of border between them with land squares on both sides of that border. In the above picture, the area with a blue border is not adjacent to the area with a green border, because of the water on the blue side of their common border. For game purposes there are only two distinct terrain types, water and land, but there are different styles of each to make the map more interesting. Game pieces can only exist on land squares. Water squares are used only to restrict movement between areas, as in the above example. If you own or know who owns a particular area, the inside of its border will be the color of the player who owns it. If it is unowned or you do not know who owns it the border color will be gray. You will only be able to see in an area if you own it or you have one or more (surviving) pieces there. To take ownership of an area, you must be the ONLY player with one or more armies there at the end of a turn.

FACTORIES

Some areas will have factories; the factory symbol will have a number on it representing the number of factories in that area. Factories are permanent fixtures on the board. Each factory produces $1 per turn for its owner, indicated by the color of the factory. In the above example, the three green factories will produce $3 each turn for their owner. If you can see into an area, the factory will be the color of the owning player, or gray if no player owns it yet. If you cannot see into an area, the factory symbol will be gray, in this case indicating that you do not know who owns it. Income not spent in a turn goes into the player's treasury which may be used later. You will always be able see all the factories (and their quantity) on the board, but you won't necessarily know who owns them, or if they are owned.

FLAGS AND SOLDIERS

When the ownership of an area changes, and you can see in the area, a flag symbol will appear to indicate the change. When a battle has occurred and armies were lost, a soldier symbol will appear, which can be clicked on to see which players lost armies, and how many they lost. Sometimes they will appear together.

The Pieces

(The Game Rules: 2 of 5)

ARMIES and SPIES

There are two types of movable playing pieces, armies and spies. Armies are represented by a face with an old fashioned military type hat, and spies by a telescope. They are the color of the owning player. The quantities of each are indicated by a number to the upper left of the piece. Armies are used to fight other armies and capture areas, Spies are used to see what is going on where you don't have armies. Spies can freely wander through enemy lines and territory, unlike armies, and are used to see into areas you do not own. Armies can be killed by other armies, spies can never be hurt. If another player has spies in an area you own you will see them, but you cannot do anything about them. Each piece (army or spy) costs $1 to build, and $1 to supply each turn.

MOVING PIECES

You may order pieces to move into any adjacent area with one exception: if you have armies in an area where at least one other player has armies, you may only move your armies out of that area into areas you already own. If you do not give an army a specific order, it is by default considered 'ordered to defend' the area. Pieces ordered to move into another area are drawn on top of an arrow indicating the move. You can freely move both your own and other players' pieces within the same area, as this does not affect the play of the game.

HOW TO MOVE PIECES

To move one piece, press the left button down on the piece, move the mouse where you want it to go and then release the left button. To move the entire quantity when there is more than one, use the middle button, or the Shift key and the left button. If there is more than one type of piece in a square, then you must move the top ones first. Use the right button, or Alt key and the left button to cancel a move into another area.

BUILDING

You may order to be built as many spies as you can afford in any area you own. Armies may only be built where you have factories, and each turn you may only build as many armies as you have factories in any particular area. Build orders are on a light blue square, think of this as a 'blueprint' to build a piece, not the piece itself.

HOW TO BUILD PIECES

Use the right mouse button in an empty square to bring up a build selection menu, or hold down the left mouse button and press the 'a' key to build an army or the 's' key to build a spy. Once you have a build order, you can use the left button to increase the quantity or the right button to decrease the quantity. Alternatively, while holding down the left button on a build order, press the 'd' key to decrease the build quantity, or the 's' or 'a' keys to increase the quantity.

DISBANDING

You may order any of your pieces to disband without restriction. If you cannot afford to supply all your units in a turn you should select which ones to disband, otherwise the computer will disband them randomly as it processes the turn. Pieces ordered to disband are shown in gray.

HOW TO DISBAND PIECES

Click the right mouse button on non-moving pieces to disband them. Click the right button again to cancel the disband. Alternatively hold down the left mouse button and press the 'd' key to disband or cancel a disband.

Combat

(The Game Rules: 3 of 5)

After all pieces move, if an area contains armies belonging to more than one player, combat occurs in that area. Spies are not involved in combat. Armies have an attack strength of one and a defense strength of two. Armies defending or moving into an area they own have a defense bonus of +1 which makes their defense strength three. Each player computes his losses based on the total attack strength of the largest single opposing force, ignoring any smaller forces which may be in the area. A player with a defense bonus loses one army for every three opposing armies in the largest opposing force. A player without a defense bonus loses one army for every two opposing armies in the largest opposing force. Combat is computed simultaneously for all players, before removing any destroyed pieces.

If two opposing forces are the same size it doesn't matter which is used, only the total attack points from either one, but only one, are important. If only two players are involved in combat then of course the other player is the largest opposing force.

EXAMPLE ONE

Red moves 2 armies into an unowned area at the same time that Blue moves 3 into that area. They will each lose one army. Neither will have a defense bonus. Blue has a total attack strength of 3, of which only 2 are needed to defeat one Red army, and Red has an attack strength of 2 which is exactly enough to defeat one Blue army.

Now there is 1 Red army and 2 Blue armies in this area which remains unowned. If no reinforcements are brought in by any player in the next turn, Red will lose his one army to Blue's 2 armies, and Blue will gain ownership of the area with no further losses.

The factory and border colors will change to blue to indicate the new owner. (This will only be visible to the Blue player unless Red has moved a Spy into the area.)

EXAMPLE TWO

Yellow is defending an area he owns with 2 armies, and Green attacks with 5 armies. Since Yellow has a defense bonus, it takes 3 Green armies to destroy 1 Yellow army. Yellow loses 1 army (Green would need 6 to destroy them both), and green loses 1 army, leaving Yellow with 1 and Green with 4. Since Green does not have a defense bonus only 2 yellow armies are needed to destroy 1 Green army.

Yellow still owns the area, as indicated by the yellow borders. If no reinforcements are brought up by either side, then on the next turn, Yellow will lose his last army, without affecting Green who will now take ownership of the area with 4 armies, needing only 3 to destroy the last Yellow army.

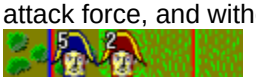
EXAMPLE THREE

In a three player example, Red defends an area he owns with 4 armies, Green moves in 3, and Blue moves in 7:

Before combat:

Player	armies	largest opponent
Red	4	(Blue) 7
Green	3	(Blue) 7
Blue	7	(Red) 4

Red sees Blue as the greatest attack force with 7 attack points. Since Red has the defense bonus, it takes 3 attacking armies to destroy each Red army. The 7 Blue armies are enough to destroy 2 Red armies. Green also sees Blue as the largest attack force, but Green does not have a defense bonus, so he loses 1 army for every 2 Blue armies, thus Green loses all his armies. Blue sees Red as his greatest attack force, and without a defense bonus, Blue loses 2 armies.



NOTES

After combat, if only one player has armies in the area, he becomes the owner. Note that if forces are fairly evenly matched, many players may have survivors. Also, two players may be fighting over an area owned by a third player, who has no armies there, and as long as both players maintain at least one army there, the third player still gets the factory income, and may still build there!

The Turn

(The Game Rules: 4 of 5)
After all players have submitted their orders, income, building, supply, moving, combat, and map updating are handled by the computer. Here is a detailed look at what happens:

1) INCOME
Each factory in an every area owned by any player produces $1 which is put into the owning players treasury.

2) PROCESS ORDERS
The computer reads human player orders, and runs the order generating programs for the computer players. Pieces ordered to move are moved, and build and disband orders are carried out.

3) BUILDING AND SUPPLY
For each player, the computer scans the map in an arbitrary order, and for each piece subtracts the build or supply cost of that piece as appropriate from the owning players treasury. If a player's treasury is zero when a piece is scanned, the piece is removed from the game.

4) COMBAT
Each area on the map is inspected to see if it contains armies belonging to more than one player. If so, combat takes place. Depending on the results, some pieces may be removed. After combat it is possible that more than one player will have surviving armies in any given area as combat does not necessarily result in only one player continuing to have pieces in any given area. A soldier symbol is placed in every area where armies were lost due to combat this turn. Players can see a soldier if either they own the area, or they lost some of the armies that caused the soldier.

5) OWNERSHIP CHECK
All areas are checked to see if they contain armies belonging to only one player. Each such area's ownership is changed to the sole player with armies in the area if they didn't already own it. Note that combat is not necessary, as one player may have no armies in or may move his armies out of an area in the same turn another player moves one in. The player moving in will now own the area. Areas continue to be owned even if they contain NO armies, until some other player takes ownership as described above. A flag symbol is placed in any area which changed ownership this turn. Players can see the flag symbol if either they own the area, or they lost some armies (and possibly ownership) there this turn. Note that if you own an area with no pieces in it, and it is taken over by another player, its border will change from your color to gray, but you will not see a flag symbol in it on the next turn.

6) WINNER CHECK
The computer checks if any player has won the game.

7) WRITE FILES
The new situation files are written for each player, and the turn processor stops.

Winning

(The Game Rules: 5 of 5)

WINNING

The game ends on turn 50, and the player with the most factories wins. Players' scores are the percentage of all factories owned at the end of the game. Due to rounding, the total scores of all players may not always add up to exactly 100.

EARLY WIN BONUS

If at any time before turn 50 a player owns at least 75% of all factories in the game, or has completely eliminated all other players, he wins immediately and scores 100, other players score nothing.

Reading the Status Bar

(The Game Interface: 1 of 2)

Just below the menu bar is a status area. This contains financial information, a killed in battle indicator or an error message.

FINANCIAL MESSAGES

Normally the status bar will contain financial information, like the example below. If it contains one of the other types of information you can click on it to restore the financial message.

Treasury $10, Income $28, Exp. $15, C.F. $13, -> $23

"**Treasury $10**" is how much money is available in your treasury to be spent.

"**Income $28**" is how much money you will get from factory production this turn. (Note: Factory production happens before combat and area ownership change, so this income is certain) Remember that each factory produces $1 per turn so the above would indicate you own a combined total of 28 factories.

"**Exp $15**" is how much it will cost this turn to build and/or support your armies and spies. Note that as you order builds and/or disbands, this number, and the ones that follow it will be immediately updated. This number indicates that you are currently supporting a combined total of 15 armies and/or spies.

"**C.F. $13**" is your Cash Flow which is income minus expenses, in other words, if this is a positive number your treasury will increase this much, and if it is a negative number, your treasury will be reduced this much. This is a good number to keep an eye on as you plan your turn.

"**-> $23**" is the projected amount of money you will have in your treasury next turn. If you proceed with your turn when this number is negative, the treasury will not actually go below zero, but the computer will randomly disband your pieces to keep your treasury at zero.

ERROR MESSAGES

When attempting to do something that is not allowed by the rules, nothing will happen. In most cases an error message will appear on the status bar in place of the financial information to indicate why you can't do what you attempted. This is a list of those messages:

Areas do not connect

You have attempted to move piece(s) to an area that is not adjacent to the area they currently are in.

Build limited by factories

You have attempted to order more armies to be built than the number of factories in that area.

Illegal exit of multiplayer-area

From an area where one or more other players have one or more armies, you have attempted to move one or more of your armies to an area that you do not own.

Can't afford

You have attempted to order a build that would cause your projected treasury next turn to be less than $0. (Disband something else, and then try again)

Can't move onto water
Can't build on water

You are not allowed to move or build pieces on water squares.

KILLED IN BATTLE MESSAGES

When you click on a gray soldier or soldier-with-flag symbol on the map, the status bar will indicate how many armies were lost in combat in that area last turn.

Menu Commands

(The Game Interface: 2 of 2)

File|Save

Saves this player's current view, and order set. This will not submit orders in a multiplayer game, but it will save them for later playing.

File|Turn

Saves this player's current view, and creates an order file. If there is only one human player, then the game will immediately proceed to the next turn, and update the player's map when finished. If there is more than one human player, then a list box will appear with the human players in the game and their ready status.

File|Return to Main Screen

Returns to main button control screen.

File|Exit

Ends program.

View|Zoom In, View Zoom Out

Change the view scale.

View|Redraw

This updates the screen. You may want this if you've moved pieces around or scrolled to a different view and move order arrows seem to be missing or have been drawn over.

View|Simplify

Tries to collect like pieces together, move unlike pieces apart, and for pieces ordered to move, tries to move them closer to the border they will cross and shorten the arrow. Try this if your orders get messy, you'll either love it or hate it. It has no effect on any existing orders and does not create any new orders.

View|Gene Editor

Puts up a window where you can modify and test your AI advisor Program.

Orders|Ask the A.I.

Runs the Genetic program associated with this player. The orders generated by this program will immediately appear on the map. You are not committed to these orders and may change or cancel them, and you may change the A.I. program and run it again or run other A.I. programs.

Orders|Cancel all orders

All builds/disbands will be canceled, and all pieces ordered to move will have their moves canceled.

Starting a Game

(Setting up a Game: 1 of 2)

PLAYER

This column indicates the color and by default the starting position on the map of each player. Red starts in the upper left, Yellow in the lower right, Green in the upper right and Blue in the lower left.

NAME

You can change the player names (up to 8 characters) if you wish. This is the filename part (not extension) that will be used for some of the player files. For humans in a multiple human game, these will be the names in the choose player listbox.

TYPE

This assigns how each player in the game will be controlled. Clicking in this field will bring up a combo box with the following choices:

> **INACTIVE**
>
> This player does not move, and defends his initial position with only his initial armies until they are defeated. If you want a game with less than 4 players, and minimum interference from the not-played players, use this.
>
> **HUMAN**
>
> This player is played interactively, for the sake of discussion we assume this means a human but this is not required. If there is only one human, then selecting **File|Turn** or pressing **F9** while playing will cause the turn to be processed and the 'human' will be immediately put back into the game. If there is more than one human, then after each turn, a list box will appear with the names of the human players in the game of which one can be selected.
>
> **AI**
>
> This player is played by the .DNA program listed in the DNA file column. In games with humans, when the humans have all submitted their orders for the next turn, the AI programs are then run, and the turn is processed.

DNA FILE

Select the DNA file to be used for order generation for this player. For human players, this will be the AI advisor program that can be run by selecting **Orders|Ask the AI** or pressing **F7**.

Network Game

(Setting up a Game: 2 of 2)

The first player must start a normal (multiple human player) game using the START NEW GAME button. The first player and any other players sharing the first player's machine to play will use the PLAY GAME button.

All players on other (networked) machines must have read/write access to the directory where the game was installed on the first players' machine and locate this using the FIND REMOTE GAME button. Then these other players use the PLAY REMOTE GAME button to play.

Once the first turn is set up, players do not need to return to the main screen, they can continue playing turns by selecting their name from the player selection list box which will appear when there are two or more human players in a game.

CHEAT WARNING

This program does not make **any** attempt to keep human players from cheating. The validity of the orders in an order file is not checked, and nothing stops a player from changing another players' orders before the next turn begins. Just as the author has more fun with computer games that do not cheat, you may have more fun if you compete only with human players that do not cheat.

On the other hand, or let's say a different perspective, since the save files are all ASCII text and pretty easy to decode, some pretty **creative** cheating is possible. However, once you start changing any files, don't count on much support even if you are a registered user!

Evolution in Cloak, Dagger and DNA

(Evolution: 1 of 5)

PROGRAM = GENE

In this game all player's game pieces move simultaneously. In order to simulate this, players do not actually move their game pieces, but submit orders for them to be moved. The game engine then handles actually moving them during the turn sequence. Thus, a player's turn consists of issuing orders for game pieces to move, build or disband. The computer players play the game by running a special program which generates the same kind of orders. These special programs can be thought of as the computer players' genetic codes or genes, and these are what the evolution engine works with.

TOURNAMENTS

Programs compete in tournaments, which is a sequence of four games played by four programs. Between each game, the programs are rotated between board positions in the same way that humans might rotate chairs in a table game. Thus each program plays once as each of the four colors (RED, YELLOW, GREEN, and BLUE) in the game. This has the effect of reducing the advantages or disadvantages any board position (color) may have. The program is treated as if it were an organism, competing with other organisms for food in each tournament. See Parameter Selection for understanding and changing food costs and rewards for playing in tournaments.

THE GENE POOL

The evolution engine manages a pool of programs, and for each program keeps track of how many tournaments it has played, and its current food supply, among other things. This pool provides the programs the evolution engine uses for tournaments, and the members (programs) of this pool are selectively eliminated and replaced using others in the pool as genetic parents.

EVOLUTION

After each tournament the programs' food supplies are checked. Any program which is out of food (zero or less) is removed from the gene pool and replaced with a new program by 'crossing' the programs of two parents. (See Parameter Selection for how the parents are chosen) Crossing is accomplished by selecting a random number of instructions from the beginning of one parent and adding to those a random number of instructions from the end of the other parent. A counter keeps track of how many instructions are added to each new program, and if this counter reaches the mutation count during the construction of a new program, (and mutation is enabled) a random instruction in the new program is 'mutated' to some valid instruction, and the mutation counter is reset.

Managing the Gene Pool

(Evolution: 2 of 5)

THE LISTBOX

Contains one line for each member of the gene pool. The gene pool is limited to 50 entries, which look like:

```
2     6      22
3*   51     -87   XYZ.DNA
4   999+    100
```

In the above example, there are 3 genes. The first, index 2, has played 6 tournaments and has 22 food points. The second, index 3, has been locked (so it will not be killed) indicated by the asterisk after the 3. It has played 51 tournaments and has a negative food point total of 87, this means it has been losing, but it cannot be killed because of the lock. It has either been loaded from the file XYZ.DNA with the ADD button, or was copied to that file with the EXTRACT button. The filename only indicates that there was a relationship at one time, since nothing stops you from editing XYZ.DNA, and this indicator will not compare them later. The last in the example, index 4, has played more than 999 tournaments, the most that is kept track of, as indicated by the plus sign after the 999. It has a food supply of 100.

PLAY>>

Press this button (or Double click the entry in the listbox) to move the selected entry in to the current players box for the next tournament.

ADD

Press this button (if there are currently less than 50 pool members) to load a new member from an external .DNA file. This will give you an open file dialog, and the file selected will be placed in an unused index in the gene pool.

EXTRACT

Press this button to save the current selection (in the listbox) to a .DNA file. You will get a file save dialog.

VIEW

When this is checked, the current selection will be disassembled into a text viewer window and updated as you change the selection.

LOCK

The button toggles a lock on the current selection. When a gene is locked it will not be replaced, even if its food goes below zero.

FILL

If the pool is not full, this button will fill the remaining slots by selecting random parents and crossing them with possible mutation. Only parents will be used that existed before the FILL button is pushed.

DELETE

Deletes the current selection from the gene pool.

CLEAR

Deletes all members of the gene pool.

HELP

Brings you to this page.

Parameter Selection

(Evolution: 3 of 5)

PLAYER SELECTION

When players (programs) are needed for a tournament, they can be manually selected with the "Play>>" button, or automatically selected by one of these methods:

NEAR

Selects gene pool programs that are near each other in the pool. The bottom and top members of the pool are considered near.

RANDOM

Selects programs from anywhere in the pool.

PARENT SELECTION

After each tournament, if new programs are created, this controls how the parents of those programs are selected. When a program is selected as parent, its food supply is reduced by the amount in the parent cost control. In this case it is temporarily allowed to exist with a negative food supply, which it MUST bring positive in its next tournament if it is to survive.

FITTEST TOURNAMENT SURVIVORS

Of any surviving tournament players, select the fittest two (most food) as the parents, if less that two can be found this way then the remaining parent(s) will be chosen by the method below:

WEIGHTED, ENTIRE POOL

For each parent program needed, a gene pool member is chosen randomly, but with a relative probability exactly equal to each programs' food supply. Thus a program with 20 food has exactly twice the chance of being a parent as a program with 10 food. Programs with a zero or negative food supply will not be selected.

FOOD

Food is the survival measure of the programs. When a tournament is played, food points are distributed to the players depending on their performance in the tournament. The total food distributed will be 100 or possibly less due to round off. The following controls select other food rules, and ranges are allowed far outside of what we believe to be reasonable for the sake of experimentation, and our own possible human error. Suggested value ranges are given for what we believe to be reasonable performance but should not be taken as gospel.

MAXIMUM FOOD STORE

This is the most food a program can store between tournaments. If a program wins more than this, it is reduced to this amount after each tournament. Allowed range is 0 to 30000. Suggested range is 20 to 200.

FOOD AT BIRTH

This is the amount that is placed in the initial food supply of a new program. Allowed range is 0 to 30000. Suggested range is 20 to 100.

COST TO BE PARENT

Each time a program is used as a parent of a new program, this is the amount the parents' food supply is reduced. Allowed range is -1000 to 1000. Suggested range is 0 to 25. Note that a negative value in this field results in additional food to the parent.

TOURNAMENT PLAY COST

This is the cost for a program to play a tournament. It is subtracted from each programs winnings after each tournament. Allowed range is -100 to 100. Suggested range is 0 to 50.

MUTATION

Whenever a new program is created, it may be subject to mutation. When a program is mutated, one of its instructions is randomly changed to some other (or possibly the same) instruction. This control allows you to control how often mutation happens, in terms of the number of instructions processed into newly created programs. For example, assume the number is 500. Each time a program is created, the number of instructions in that program are counted. When the counter total for all programs created reaches 500, the program just created is mutated, and the counter is reset. To have NO MUTATIONS, set this to 0.

Tournament Results

(Evolution: 4 of 5)

After the completion of tournaments during the run of the program, this screen will display statistics regarding the most recently completed tournament, which will look something like:

Tournament 10780

Gene	28	29	30	32	
Red	100	0	100	0	50
Yellow	0	0	100	0	25
Green	0	0	0	0	0
Blue	0	0	100	0	25
Score	25	0	75	0	
Food	0	55	100	-25	

Gene 28 = 30(10) X 29(5)
Gene 32 = 29(7) X 30(12) M@3

At the top is the tournament number for the current gene pool. This is followed by four columns of information relating to each of the four genes which participated in the tournament. At the top the gene index numbers head the columns, this is followed by the score of each gene as it played each of the four colors (board positions) during the tournament. In the above example, looking at the column for gene index 28, it scored 100 in the game where it played the Red board position, and 0 in the other three games. Gene 28's tournament score was 25, which is the average of the scores from the four games played, and its remaining food after this tournament was 0.

Following the results table, statistics on gene replacement will appear if any genes were replaced. In the above example, gene 28 was replaced by a cross of genes 30 and 29, with the first 10 instructions coming from the beginning of gene 30 and then 5 instructions coming from the end of gene 29, with no mutations.

Gene 32 was replaced, with the first 7 instructions coming from the beginning of gene 29 and then 12 instructions from the end of gene 12. The M@3 indicates that the new gene 32 had its third instruction mutated, changing it to some (legal) random instruction, including the possibility of the same one.

This page will be updated at the end of each tournament, so if you are examining this page during a run, be prepared for it to be updated with new results. You might want to run only one tournament at a time or press the PAUSE button (the RUN button changes into this when pressed) while you are looking at this page.

Running the Evolution Engine

(Evolution: 5 of 5)

CURRENT PLAYERS

This list box contains the players either playing or selected to play the next tournament. You can select players into this box with the "Play>>" button or Double Clicking from the "Gene Pool" tab-page. Double click an entry in this list box to remove it. If less than 4 entries are here when the RUN button is pressed, it will be filled automatically as specified in PLAYER SELECTION on the "Parameters" tab-page.

STATUS AREA

Indicates tournament number for this gene pool run, and status bars for each of the 4 games in the tournament during a run.

RUN MODE

ONE TOURNAMENT

Select this to play only one tournament when the RUN button is pressed, or select this while a tournament is running to stop when it is done.

CONTINUOUS PLAY

Select this to play continuously when the RUN button is pressed. If you are planning to run a large number of tournaments, you may want to minimize the Evolution Lab screen. On machines with slower graphics cards this may slightly improve performance.

ENGINE

CONTEST ONLY

Select this to disable all evolution functions, and just play tournaments, adjust game counts and food supply. Programs with negative food will not be deleted in this mode.

ALLOW EVOLUTION

Select this to allow (non-locked) pool members to be deleted and replaced by evolution functions.

RUN (PAUSE)

Press this to start playing one or more tournaments. Any players selected into the Players list box will be selected to play, and any empty slots will be filled with other pool members according to the PLAYER SELECTION control on the PARAMETERS tab-page. There must be at least four members of the gene pool for this to work properly, since duplicates will not be selected automatically. The RUN button will become a PAUSE button which you can press to temporarily stop the current tournament. Press again to resume.

Concepts

(Gene Editing: 1 of 4)

Each genetic program playing the game or used to advise a human is represented by a small program in a language specifically designed for this task. The language has no loops or branching, it starts at the beginning and runs to the end. There is a limit of 25 instructions a program may have, including the required Halt at the end. The program generates build, disband and move orders for a player's pieces in the game. If the program does not specifically give an order to a piece then the default orders for that piece is to defend which is the same as doing nothing. A program generated randomly in this language is guaranteed to terminate in a finite time, but may not generate any orders, in this case the pieces would just sit and defend until the end of the game or until they are destroyed by other players. A program only considers one turn in isolation, thus there is no provision to store information to be used in a later turn.

Architecture

(Gene Editing: 2 of 4)

AREA REGISTERS

There are 4 registers: R0, R1, R2 and R3 for each area in the game. These can have independent values for each area and are processed in parallel. For example, if an instruction adds five to R3, it will add 5 to the R3 register in every area. These registers can hold values from -32768 to 32767.

At program start, area register R0 is 1 for all areas, and R1, R2 and R3 are 0 for all areas. In this document Rx, <Rdst>, and <Rsrc> will be used to indicate any of the area registers.

Most instructions which can change R1 (they will have an R1 in a field named <Rdst>) are conditional on the value of R0 in the same area being greater than zero. The Notes section for each individual instruction will indicate any exceptions. Thus, for most instructions, if R0 is 0 (or negative) in a given area, then an instruction that modifies R1 would be skipped for that area. In any other areas where R0 is greater than zero, the instruction modifying R1 would be executed. R2 and R3 are general purpose area registers and have no special restrictions.

The area registers can be thought of as if they represent elevations in each area, where a higher elevation is owned and more protected, and a lower elevation is enemy owned and desirable. The move, build and disband instructions will use these 'elevations' to decide where to move, build and/or disband and in what order.

SPECIAL REGISTERS

There are two special purpose registers used to set thresholds for the build and disband instructions: "Treas" and "Cf" which unless changed by the Set instruction, default to 0. There are only one each of these registers, not one for each area.

Instruction Set

(Gene Editing: 3 of 4)

Halt	Stop program execution.
Set	Put a value in one of the specific purpose registers.
Mov	Put a value or piece count in each of one Rx.
Add	Add a value or piece count to each of one Rx.
Sub	Subtract a value or piece count from each of one Rx.
Mul	Multiply each of one Rx by a value or piece count.
Div	Divide each of one Rx by a value or piece count.
Avg	Average each of one Rx with all its immediate neighbors.
Distf	Add a 'distance to front' value in each of one Rx.
Move	Generate move orders into any areas
Build	Generate build orders
Disband	Generate disband orders

Halt

(Instruction Set: 1 of 12)

Purpose

Stops program execution. No instructions are executed after a Halt. Any program with more than 25 instructions will have the first 24 instructions run, and then it will halt, regardless of whether the 25th instruction actually is a Halt.

Syntax

```
Halt
```

Example

```
Build army 20
Halt
```

Set

(Instruction Set: 2 of 12)

Purpose

Put a value in one of the specific purpose registers.

Syntax

Set Treas <Ttest>
Set Cf <Ctest>

<Ttest> range is 0..255
<Ctest> range is -128..127

Notes

These registers default to a zero value. See the Build and Disband instructions for information on how these values are used.

Examples

Set Treas 10
Set Cf 4
Set Cf -2

Mov

(Instruction Set: 3 of 12)

Purpose

Move a constant value, a value from another register, or piece count into an area register set. This replaces the previous value in that register which is lost.

Syntax

```
Mov <Rdst> <data>
Mov <Rdst> <Rsrc>
Mov <Rdst> my army
Mov <Rdst> my spy
Mov <Rdst> my factory
Mov <Rdst> my area
Mov <Rdst> notmy army <default_value>
Mov <Rdst> notmy spy <default_value>
Mov <Rdst> notmy factory
Mov <Rdst> notmy area
```

<Rdst>,<Rsrc> are one of:	R0, R1, R2 or R3
<data> is a constant:	-128 to 127
<default_value>is:	0 to 31

Notes

The forms using the "my" keyword will place the appropriate quantity of game pieces present into the <Rdst> registers, or 0 if none are present in a particular area. The form "my area" will use 1 in owned areas, and 0 in all other areas, and "notmy area" will be the reverse, or 1 in unowned areas, and 0 in owned areas.

The forms using "notmy army" and "notmy spy" will place the quantity of the largest single opposing force (of the specified type of piece) into the <Rdst> registers, or, if this value is not known, <default_value> will be placed in the <Rdst> registers.

When "my factory" or "notmy factory" is used, the number of factories of appropriate ownership will be placed in <Rdst>, or 0 if no factories.

Remember that instructions using R1 as <Rdst>, will not be executed in areas where R0 is 0 or less.

Examples

Mov R0 5

Move 5 into all R0 area registers.

Mov R1 R2

In all areas where R0 is greater than 0, copy the value in register R2 into R1. Note that since R1 is the <Rdst> register, it can be modified by this instruction, thus the instruction is only executed in areas where R0 is greater than 0.

Mov R2 R1

In all areas, copy the value in register R1 into R2. Note that since R1 is not modified, (it is <Rsrc> not <Rdst>) this instruction is not subject to the R0 restriction.

Mov R2 my army

In each area, move the number of the player's armies present into R2.

Mov R0 my area

Place 1 in R0 for areas owned by the player, place 0 in R0 for areas not owned. Until R0 is modified, instructions following this that would modify R1 will now only be executed in areas owned by the player.

Mov R1 notmy area

In all areas where R0 is greater than 0, place 1 in R1 for areas not owned by the player, place 0 in R1 for areas owned.

Mov R2 notmy army 5

In all R2 registers, place the number of armies of the single largest opposing force in R2, or in any area where this information is not known, place 5.

Add

(Instruction Set: 4 of 12)

Purpose

Add a value or piece count to an area register set.

Syntax

Add <Rdst> <data>
Add <Rdst> <Rsrc>
Add <Rdst> my army
Add <Rdst> my spy
Add <Rdst> my factory
Add <Rdst> my area
Add <Rdst> notmy army <default_value>
Add <Rdst> notmy spy <default_value>
Add <Rdst> notmy factory
Add <Rdst> notmy area

<Rdst>,<Rsrc> are one of:	R0, R1, R2 or R3
<data> is a constant:	-128 to 127
<default_value>is:	0 to 31

Notes

The forms using the "my" keyword will add the appropriate quantity of game pieces present into the <Rdst> registers, or 0 if none are present in a particular area. The form "my area" will add 1 in owned areas, and 0 in all other areas, and "notmy area" will be the reverse, or 1 in unowned areas, and 0 in owned areas.

The forms using "notmy army" and "notmy spy" will add the quantity of the largest single opposing force (of the specified type of piece) to the <Rdst> registers, or, if this value is not known, <default_value> will be added to the <Rdst> registers.

When "my factory" or "notmy factory" is used, the number of factories of appropriate ownership will be added to <Rdst>, or 0 if no factories.

Remember that instructions using R1 as <Rdst>, will not be executed in areas where R0 is 0 or less.

Examples

Add R2 5
Add 5 to all of the R2 registers.

Add R1 R3
in each area, add R3 to R1 if R0 in that area is non-0.

Sub

(Instruction Set: 5 of 12)

Purpose

Subtract a value or piece count from an area register set.

Syntax

Sub <Rdst> <data>
Sub <Rdst> <Rsrc>
Sub <Rdst> my army
Sub <Rdst> my spy
Sub <Rdst> my factory
Sub <Rdst> my area
Sub <Rdst> notmy army <default value>
Sub <Rdst> notmy spy <default value>
Sub <Rdst> notmy factory
Sub <Rdst> notmy area

Notes

The forms using the "my" keyword will subtract the appropriate quantity of game pieces present from the <Rdst> registers, or 0 if none are present in a particular area. The form "my area" will subtract 1 in owned areas, and 0 in all other areas, and "notmy area" will be the reverse, or 1 in unowned areas, and 0 in owned areas.

The forms using "notmy army" and "notmy spy" will subtract the quantity of the largest single opposing force (of the specified type of piece) from the <Rdst> registers, or, if this value is not known, <default_value> will be subtracted from the <Rdst> registers.

When "my factory" or "notmy factory" is used, the number of factories of appropriate ownership will be subtracted from <Rdst>, or 0 if no factories.

Remember that instructions using R1 as <Rdst>, will not be executed in areas where R0 is 0 or less.

Examples

Sub R3 notmy army 5
Subtract opponents army count from R3, if player can't see this area, subtract 5.

Sub R3 5
Subtract 5 from all R3 registers.

Mul

(Instruction Set: 6 of 12)

Purpose

Multiply an area register set by a value or piece count.

Syntax

Mul <Rdst> <data>
Mul <Rdst> <Rsrc>
Mul <Rdst> my army
Mul <Rdst> my spy
Mul <Rdst> my factory
Mul <Rdst> my area
Mul <Rdst> notmy army <default value>
Mul <Rdst> notmy spy <default value>
Mul <Rdst> notmy factory
Mul <Rdst> notmy area

Notes

The forms using the "my" keyword will multiply <Rdst> by the appropriate quantity of game pieces present, or 0 if none are present in a particular area. The form "my area" will multiply <Rdst> by 1 in owned areas, and 0 in all other areas, and "notmy area" will be the reverse, or 1 in unowned areas, and 0 in owned areas.

The forms using "notmy army" and "notmy spy" will multiply <Rdst> by the quantity of the largest single opposing force, (of the specified type of piece) or, if this value is not known, <Rdst> will be multiplied by <default_value>.

When "my factory" or "notmy factory" is used, <Rdst> will be multiplied by the number of factories of appropriate ownership, or 0 if no factories.

Remember that instructions using R1 as <Rdst>, will not be executed in areas where R0 is 0 or less.

Example

Mul R2 notmy factory

Multiplies the value in each R2 by the number of factories in each unowned area. In owned areas and areas with no factories, R2 is multiplied by 0.

Div

(Instruction Set: 7 of 12)

Purpose

Divide an area register set by a value or piece count.

Syntax

Div <Rdst> <data>
Div <Rdst> <Rsrc>
Div <Rdst> my army
Div <Rdst> my spy
Div <Rdst> my factory
Div <Rdst> my area
Div <Rdst> notmy army <default value>
Div <Rdst> notmy spy <default value>
Div <Rdst> notmy factory
Div <Rdst> notmy area

Notes

The forms using the "my" keyword will divide <Rdst> by the appropriate quantity of game pieces present, or 0 if none are present in a particular area. The form "my area" will divide <Rdst> by 1 in owned areas, and 0 in all other areas, and "notmy area" will be the reverse, or 1 in unowned areas, and 0 in owned areas.

The forms using "notmy army" and "notmy spy" will divide <Rdst> by the quantity of the largest single opposing force, (of the specified type of piece) or, if this value is not known, <Rdst> will be divided by <default_value>.

When "my factory" or "notmy factory" is used, <Rdst> will be divided by the number of factories of appropriate ownership, or 0 if no factories.

Whenever this instruction would attempt to divide by zero, the largest representable value will be placed in <Rdst> which is 32767.

Remember that instructions using R1 as <Rdst>, will not be executed in areas where R0 is 0 or less.

Examples

Div R1 2

In every area where R0 >= 0, divide R1 by 2.

Div R3 R2

In every area, R3 becomes the value of R3 divided by R2. Exception: anywhere that R2 is zero, 32767 will be placed in R3.

Avg

(Instruction Set: 8 of 12)

Purpose

Average each <Rdst> with all its immediate neighbors.

Syntax

Avg <Rdst> <passes>

<Rdst> is R0, R1, R2 or R3
<passes> is 0..3

Notes

If <passes> is 0 then nothing happens.

Each node becomes average of itself and all its neighbors. This process is repeated <passes> times. If R1 is averaged, only areas where R0 > 0 will be modified, however all areas that are modified will use the values in ALL neighbors, regardless of the neighbors' R0 value, to compute the new value.

Example

Avg R2 3

DistF

(Instruction Set: 9 of 12)

Purpose

Add a 'distance to front' value in each area's <Rdst>.

Syntax

DistF <Rdst>

Notes

In all the players' owned areas, the shortest number of moves it would take a spy (spies have no movement restrictions) to enter an unowned area is **added** to Rdst. Nothing is changed in unowned <Rdst>'s. This is a simple way to generate a slope leading to the front lines.

Example

DistF R2

Move

(Instruction Set: 10 of 12)

Purpose

Generate move orders for some or all of a particular type of piece.

Syntax

Move army <weight>
Move spy <weight>

<weight> is -128 to 127

Notes

The move instruction uses R1 as an elevation, however it does not generate orders for pieces in areas where R0 <= 0. **R1 exception:** this instruction may generate orders to move pieces into areas where R0 <= 0, in which case, it is allowed to change R1 in those areas.

Starting with the area having the highest value (elevation) in R1, and working down, this instruction examines pieces of the type specified and considers moving them to lower elevations. Each piece examined is temporarily 'removed' from the board by adding <weight> to the R1 of the area it is in. Then if at least one neighboring area has a lower elevation, the piece is given orders to move to the lowest of those neighbors, and that neighbor's R1 has <weight> added to it. Each remaining piece in the same area, if any, is examined in the same way using the R1 values just modified by the previously moving piece. If there is no lower neighbor, <weight> is added back to the current area, any remaining pieces in this area are not considered further, and the next lower elevation with pieces of the type specified are considered for moving. Note that <weight> is allowed to be negative, which may or may not be useful.

Examples

Move Spy 5
Move Army 80

Build

(Instruction Set: 11 of 12)

Purpose

Possibly generate build orders for a specified type of piece. Build prefers to generate build orders at lower R1 values, which tend to be closer to enemy pieces and areas.

Syntax

Build army <weight>
Build spy <weight>

<weight> is -128 to 127

Notes

Build creates a list of areas by sorting all areas owned by the values in R1. It will only consider areas that have R0 >= 0. Starting with the area having the lowest R1 and working upwards, build may generate a build order for a spy or army. Before each attempted build, this instruction checks the Treas and Cf registers which can be changed by the Set instruction. Build will only build a piece, if, after the build, the players treasury will be greater than or equal to the value in the Treas register and the players Cash flow (after the build) will be greater than or equal to the value in the Cf register. If either of these tests fails, then the build instruction passes control to the next instruction, possibly without having generated any build orders.

When build decides to generate a build order for a piece, it will add the value in the <weight> parameter to the area given the build instruction, and re-sort this area back in the list before attempting the next build. Thus a large weight will tend to spread out build orders over a number of areas, and a small weight will tend to build a larger number of pieces in a few areas.

Examples

Build army 20
Build spy 100

Disband

(Instruction Set: 12 of 12)

Purpose

Possibly generate disband orders for a particular type of piece. Disband prefers to generate disband orders at higher R1 values, which tend to be farther away from enemy pieces and areas.

Syntax

Disband spy <weight>
Disband army <weight>

Notes

Disband creates a list of areas by sorting all areas with the specified type of piece by the values in R1. It will only consider areas that have R0 >= 0. Starting with the area having the highest R1 and working down, Disband may generate a disband order for a spy or army. Disband checks the Treas and Cf registers which are set by the Set instruction. Disband will only disband pieces as long as the player's treasury is less than the value in the Treas register OR the player's Cash flow is less than the value in the Cf register.

When Disband decides to generate a disband order for a piece, it will subtract the value in the <weight> parameter from the area given the disband instruction, and re-sort this area back in the list before attempting the next disband. Thus a large weight will tend to spread out disband orders over a number of areas, and a small weight will tend to disband a larger number of pieces in a few areas.

The disband instruction is useful in controlling how pieces are disbanded when this is necessary. Without this instruction, if a situation developed where the treasury could not cover the current supply costs, the necessary pieces would be automatically disbanded at random by the turn engine.

Examples

Disband army 20

Using the Gene Editor

(Gene Editing: 4 of 4)

You must be playing a game (as a human) to get to the Gene Editor. Select **VIEW|Gene Editor** from the menu. You will get a window that shows your current 'advisor' program. You can edit this program, and when you select **Run!** from the menu it will compile and if there are no errors, the program will be run, immediately generating orders for your game pieces. If there are errors, a '**?**' will appear at the left of any line with an error and another '**?**' will appear immediately after the token or symbol that was not understood. You can fix the error and then hit **Run!** again, ignoring the '**?**'s which will be automatically removed when there are no errors. Once the program has run successfully, you can select the **Run Info** tab to see information on the game registers as modified by each instruction.

You can use this to learn the instruction set. Play a game to turn 4 or so, enough that you own a few areas, and can see more than one opponent. Now, using the gene editor, write very small programs, one line at first, run them, and observe the output on the Run Info page. Following each instruction that modifies a register set is a list of the values of each register. For example, write and run the one line program "DistF R2". The output will show for each area, how far it is from the closest unowned area.

Also on the menu are standard **File|Open...**,**File|Save as...**, and **File|Close** selections. Use the open and save to save this as a .DNA file.

Once you have modified the .DNA file in any game, it will automatically be saved with an ADV extension (in the directory where you installed the program) as an advisor file. If you were playing GREEN, then it would be saved as GREEN.ADV. When the next turn is loaded, if this file exists, it will be loaded instead of GREEN.DNA. This way you do not have to save and reload each turn, and the original GREEN.DNA is not modified. After the game, if you want to keep your new .ADV file you should rename it to something with a DNA extension, using DOS or the File Manager. The format of the .DNA and .ADV files are identical. Advisor (.ADV) files will be deleted when the next game starts.

A Note from the Author

(End: 1 of 2)

This program was largely inspired by my interests in games and the (computer science) field of Genetic Algorithms, and a desire to combine them in an interesting way. A small amount of inspiration is from my frustration with some computer strategy games where the computer players play by a different set of rules than the humans, in other words, they have to "cheat", in order to play an interesting game.

Of course, if a computer game company were to put the time and resources necessary into every game to make the computer opponents smart enough to play by the same rules, some of those games would have to be pretty expensive! That company probably wouldn't stay in business very long, unless they could find a large market for games costing thousands of dollars each! Even for computer scientists, some games can present very difficult problems. As I write this, (September 1995) I have yet to heard of a Chess program that consistently beats the best human Chess players, although I have heard that a Backgammon program based on neural networks has caused some reconsideration of at least one of the accepted standard opening moves.

Occasionally the author of a research paper in the Genetic Algorithm field will document a surprise, usually some success of evolution that was unexpected. During the development of this program I experienced a surprise, but of a slightly different nature. As the computer players evolved to the point where some could occasionally beat me in a game, I realized that this was a good thing. Normally I'm not especially pleased when the computer wins a game, but in this case, losing became a measure of success!

Probably the biggest non-programming challenge has been trying to find a reasonable balance between a game that is complex enough to be interesting (to humans) for at least a few games, and yet simple enough that a fairly small program with a limited instruction set (representing the genetic code) could play a reasonably competent opponent. With a more complex instruction set or a larger program, I was worried that a home computer would not be capable of evolving anything interesting in a short time span, say a few hours, or overnight at the most. As I write this probably close to a total of half a million evolution tournaments have been played on various machines with various parameters, and sometimes twenty thousand or more tournaments can go by with no obvious improvement. Have we seen the best possible players already, or are the best we've seen so far only simpletons compared to what will come? Only time and more users (hopefully registered ones!) can tell.

I would like to hear what you think about this program even if you haven't decided to register! And even if the computer never beats you. (Try doing nothing on your first turn.) If you write or evolve some really super opponents, please send them along, and challenge your friends come up with better ones!

Please understand that unless a miracle occurs, support of this program will be a part-time effort, and if you have a question or desire a response, it could take some time for me to get back to you, which I will try to do. Of course for practical reasons, registered users will be put in the queue ahead of everyone else!

I hope you have as much fun playing/experimenting/learning with this program as I had creating it!

Don O'Brien

Oidian Systems
P.O. Box 700365
San Jose, CA 95170-0365

CIS: 71702,2255 (71702.2255@compuserve.com)

Bibliography

(End: 2 of 2)

Brooks, R.A., P. Maes, *Artificial Life IV* (proceedings)
MIT Press, 1994

Dawkins, R. *The Blind Watchmaker*.
Norton, 1987

Dawkins, R. *The Selfish Gene*.
Oxford, 1976

Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.*
Addison-Wesley, 1989

Langton, C.G., editor *Artificial Life* (proceedings)
Addison-Wesley, 1989

Langton, C.G., C.Taylor, J.D. Farmer, & S. Rasmussen, editors *Artificial Life II* (proceedings)
Addison-Wesley, 1992

Langton, C.G., editor *Artificial Life III* (proceedings)
Addison-Wesley, 1994

Ray, T.S., *An approach to the synthesis of life*.
In: Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen, S., editors Artificial Life II, pp. 371-408. Addison-Wesley 1992