# GPRobots and GPTeams - Competition, co-evolution and co-operation in Genetic Programming

**Conor Ryan**
Computer Science Dept.
University College Cork
Ireland

### Abstract

This paper presents two simulations, ***GPRobots*** and ***GPTeams***. GPRobots is a new, competition oriented benchmark for GP, where control programs are evolved for robots. These are then executed in a simulated *real time* environment, where execution time for instructions varies with instruction complexity. The competitive nature of GPRobots allows the direct comparison of different evolutionary approaches, using two or more *best-of-run* individuals from various simulations.

GPTeams, the second simulation in this paper, introduces the idea of using GP to produce *event driven* programs, using a novel technique where a population of main control programs are co-evolved with the callback functions they use. The callback functions, although competing against each other on an evolutionary scale, co-operate in teams organised by the main program at a local level.

The experiments produce a wide array of behaviours from individuals, from pacifist to violent, from parasitic to general.

## Introduction

Games of competition are often used as a test of programming skill (Dewdney 84)(Rognlie 93)(Timin 95)(Schick 94). These comparisons usually take the form of a competition between different programmers who try to outwit each other in the game. Often, the inner workings of the programs entered into these competitions are kept secret and the source is not publicly available. Under certain circumstances an optimal solution is known, as in the well known Prisoner's Dilemma game, so it is not really useful to directly compare different development methodologies. This paper develops a similar test for GP, where individuals produced by different evolutionary approaches may be tested against each other.

Competition, particularly in the form of co-evolution, (Siegel 94) (Koza 92) (Jannink 94) in GP is not uncommon, although competition between individuals in a single population has also been investigated (Reynolds 94a). Invariably, these are one-on-one competitions[1], but GPRobots allows more than two individuals in a tournament, and, as points are awarded for longevity relative to the others in a tournament, it is possible for a number of different strategies to emerge, e.g. a pacifist robot who avoids combat may survive longer than a very aggresive robot who constantly attacks.

In scenarios where individuals are pitted against each other, usually a move is determined for each, and then executed, implying that each move takes the same length of time to execute. GPRobots differs from this, in that all instructions take an argument, e.g. how far to move forward, how many degrees to turn left etc.. We share the view of (Reynolds 94b) that more complex or longer moves should take more than one time unit to execute.

A relatively new programming paradigm is that of *event driven programming*. Event driven programming differs from most paradigms in that instead of writing a sequential program, a programmer writes functions to deal with particular events that may (or may not) happen during the running of a program. Each function is registered with an *event handler* which calls each function as appropriate. It is possible to prioritise these events, so an event with a higher priority may interrupt a lower priority event.

In practice, event driven programming is really interrupt driven programming at a user rather than hardware level, in that events can interrupt the normal flow of a program, and, in some cases, pre-empt each other.

## GPRobots Tournaments

Most previous work with GP involving competition between individuals has been concerned with one-on-one competitions. GPRobots conducts its tournaments between two *or more* robots - three robots in the simulations presented here. Robots score points for the amount of other robots killed while they are alive. The

[1] Although some researchers, notably (Siegel 94) and (Hillis 91), pit a single individual from one population against several from another, each individual from the second population fights the first individual singly.

arena in which tournaments take place measures 75x20 squares, with robots taking up 3x3 squares.

Each robot starts with 100 energy units. Units are lost by shooting at other robots, by being hit by another robot's shot, or by crashing into another robot. Energy is *not* lost by colliding with a wall of the arena.

The term *round* is used to denote one time step, and the term *turn* to denote the action a robot wants to take. As will be seen, some turns require multiple rounds to achieve.

### Robot Actions

As stated above, it is common practice when evolving control programs with GP (such as the well known *Artificial Ant* problem(Koza 92)), that the actions involve the robot moving ahead one square, going back one square, turning left 90° etc. To add realism to the simulation, robots in GPRobots are allowed turn facing any angle between 0° and 360°, and that robots who are travelling forward several squares should accelerate.

Due to pressures of space, and to allow proper discussion of the results, only a very brief account of the workings of the robots will be given.

The most comprehensive work in this area with GP has been that of Reynolds(Reynolds 94a) and his is the only work to permit robots to be continuous in orientation. GPRobots adopts this approach, and also assumes that robots accelerate if they are travelling continuosly in the same direction. All but one of the robot instructions are of the form: **(action argument)**, and all return a value indicative of the success or otherwise of the instruction.

Robots may turn **left** or **right**, and may move **ahead** or **reverse**, and, of course, may **fire**. Left and right take an argument of degrees to turn, while ahead and reverse the number of squares to move. The fire instruction doesn't take any arguments.

### Robot sensors

Robots have available a number of sensors to them which provide information about the arena. These include a sensor which reports a robot's involvement in a collision and a sensor which reports the robot being struck by a missile. These sensors return the number of degrees which the robot must turn to face either the other robot involved in the collision or the robot who fired the offending missile. Robots can see other robots who are directly in their line of vision, and can determine how far away they are.

For the purpose of evolving robots, conditional commands which directly access the contents of the sensors such as **(IF-Collide $x$ $y$)** could be designed. It is the view of the author that functions such as these are not an integral part of the robot structure, no more than say, a mathematical function would be, and for this reason, are not considered to be "hardwired" into the robot.

This is not to say that the programs controlling the robot cannot use these, or, indeed, any other reasonable functions that an implementor wishes to use.

## Fitness Function

One of the reasons that games such as this are so popular as a test of programming skill and strategy is that there is usually no obvious optimal strategy[2]. While this makes the benchmark more interesting and useful, it complicates the fitness function when it comes to evolution.

One approach is to hand code a few expert robots, and use these to test the *best-individual* produced by a GP run(D'haeseleer 94)[3]. However, this would seem to suggest that the ability of GP to produce individuals is determined to some extent on the ability of the implementor to produce hand coded solutions to the very problem that is to be solved, a situation we would rather avoid.

Given that there is no obvious fitness measurement, the only other choice is to measure an individual's fitness relative to other individuals. Problems such as this involving competition between individuals lend themselves to this sort of measurement to some degree. Only to some degree because, although it is easy to compare $k$ individuals, where $k$ is the number involved in a tournament, it is another matter to compare $N$ individuals, where $N$ is the size of the population.

A variety of these methods were reported on by (Reynolds 94b) and are summarized in the table below.

| Competition | Matches per ind. |
|---|---|
| New versus all | $n-1$ |
| New versus several | k |
| Knockout tournament | $\log_2 n$ |
| New versus best | 1 |
| New versus new | 1 |
| New versus neighbour | 1 |

This paper uses a method which employs both the *New versus several* (Reynolds 94b) and the *New versus neighbour* (D'haeseleer 94). Individuals live in a one dimensional neighbourhood, and are chosen in groups of $k$ for tournaments.

Individuals in this particular simulation of GPRobots are arranged on a one dimensional toroidal grid, in a manner similar to (Collins 92). Rather arbitrarily it was decided to implement overlapping demes each with three individuals. This

[2]While this is not strictly true in the case of the Prisoner's Dilemma game, it is the modelling of games such as those mentioned in the introduction that we are most concerned with.

[3]D'haeseleer and Bluming's work also used the problem of robots fighting each other in an arena, but they were more concerned with the effects of locality than the evolution of interesting behaviours as is the case in this paper

resulted in each robot being involved in three fights, with each robot in a fight being awarded one point for each robot killed while they are still alive. Their fitness is the sum of their points over three games. A robot who comes last in all three fights amasses three points.

Crossover is implemented by using these demes, in a similar manner to (Collins 92), except, that if the individual to be replaced has the highest fitness in the deme, then that individual is instead selected for reproduction.

The reader should be aware that all implementation decisions, such as those involving the spatial structure, points allocation, tournament size etc. are peculiar to this attempt to evolve interesting robots. Other approaches could yield vastly different results. In fact, it is the hope of the author that other implementations be examined, and it would cause neither surprise nor disappointment if a different implementation yielded superior results.

## Preliminary Results

Normally, a GP run reports a *best-of-run* individual who performed better than any other, and when reporting on results, one can often quote statistics showing the percentage of successful runs / average generations to success etc. Unfortunately, the success or otherwise of a run in this case isn't quite so clear-cut. Firstly, there is no absolute "solution" to the problem. The second problem is that the performance of an individual can be measured only relative to nearby individuals. Both these problems were also encountered by (D'haeseleer 94), mentioned above, and their solution was to test potential *best-of-run* individuals against a group of handcoded individuals.

In this paper, we would rather not rely on the ability of the implementor to gauge the quality of the results, and so the discussion below will concentrate on the more interesting individuals who appeared and the strategies they adopted. These individuals were chosen for their longevity, as an individual who was reproduced several times is clearly employing some sort useful strategy. Individuals who were rated "interesting" using this criterion were then set fighting against each other, and in this way, a *best-of-run* individual was discovered.

### Sloths, Triggers and Spinners

There were two surprising aspects of the results. The first was how few of the successful individuals ever utilised the **AHEA** or **REV** commands, most individuals being content to remain stationary, turning in one postion whilst searching for opponents. The second surprise was how many of those robots selected as interesting actually did nothing. The robots had available to them a number of mathematical functions, and it wasn't uncommon for these individuals to consist of a simple mathematical expression, such as

**( + ( * 5 2 ) 1 )**

These individuals, dubbed *Sloths*, often appeared near each other in the population structure, very often on either side of another common individual, which we name *Trigger*, who simply fires every turn. Unless a Trigger was very fortunate, it usually used up all its energy firing and died quite early, while the Sloths simply sat out the hostilities until the time limit for a fight ran out. The Sloths then, operate as a form of parasite, living off the behaviour of the Triggers, who, acting as "hosts" of sorts, provided them with an easy life.

It is reasonable to assume that with a bigger deme size, Sloths would find it more difficult to find the right conditions to allow their survival. For the sake of consistency though, this paper will not try an experiment with a larger deme.

Rather reassuringly, another type of individual also appeared, who, while not quite as numerous as the Sloths, was more successful. These individuals are known as *Spinners*. The common trait in Spinners is that they turn slowly, scanning the grid as they do so, until they see another robot. Upon seeing another robot they fire continuously until the robot is either dead or runs away. These individuals are normally of the form

**(IF-See (FIRE) (LEFT 2))**

Spinners represented the best strategy, and, in direct competition with a Sloth, a Spinner would almost always win. In certain cases, however, in a competition involving two Spinners and one Sloth, the Sloth won the competition because the two Spinners found and attacked each other.

### The best of run

To find the best-of-run individual, several (up to ten) individuals were selected from a run based solely on their longevity. These ten were then pitted against each other in another tournament, with the winner of that being declared the *best-of-run* individual. This then automated the selection of such an individual from a run.

The best individual to appear was called a *Smart Spinner* and was as follows

**(IF-See (FIRE) (LEFT (IF-Missile (bearing_of_enemy) 5)))**

Like its ancestor, the Smart Spinner turns slowly, looking for enemies, but, if the Smart Spinner is hit by a missile, it turns to face the offender. In contrast, the ordinary Spinners do not react to being shot, instead continue turning and searching.

It is suggested that different approaches to evolving robots could be compared in a similar manner, with perhaps the best two or three from each approach being entered into a tournament with each other.

Only a small, albeit representative, selection of individuals are presented above. Other individuals also

appeared, such as *Panickers* who turned and fled upon being shot or involved in a collision, or *Corners*, individuals who backed themselves into a wall or corner and tried to wait out the fight there.

## GPTeams - Evolving event driven programs

A relatively new programming paradigm is *Event Driven Programming*, used extensively when programming in GUI environments such as Xview, Motif etc. Events are stimuli which cause the program to take notice, and such notice is given by an *event handler*. Events may be a user clicking a button on a menu, a user closing a window or anything else that may affect the operation of a program.

Generally, a function, a *callback function*, is written for each event and then registered with the event handler. As events occur the appropriate function is called. In the case of two events happening at once, or an event occurring while the callback function of another event is executing, the events may be queued, or, in the case of prioritised events, the highest priority event may interrupt other events.

As mentioned in the introduction, event driven programming is really interrupt driven programming at a higher level. As such, there are an enormous number of real world applications, from washing machines to microproccesors, that can utilise this type of programming. It is suggested that Genetic Programming is suitable for the automatic generation of event driven programs, and the remainder of the paper shows how, using a combination of competition, co-evolution and co-operation, GP can evolve event driven programs.

### Evolving Events

Multiple co-operating populations have been used before (Ryan 94a) (Ryan 94b) (Ryan 95) (Siegel 94), and in this case, two populations are used. One population contains individuals who will be used as callback functions(CBs), while the other will contain the main programs(MPs) which decide when it is appropriate to call functions provided by the CBs. Each MP is made up of a number of *event registers* which register an event with the event handler. Each registered event is registered as follows:

**(Reg_Event *Condition Priority Code*)**

Where *Condition* is the condition under which the callback function is to be executed, *Priority* is the priority of this event, and *Code* is a pointer to the callback function.

Evolving event driven programming involves both competition and co-evolution. Individuals from MP compete with each other by organising individuals from CB into co-operating teams. The co-operating teams then compete against each other. Each MP robot contains pointers into the CB population to call individuals when appropriate. The individual pointed

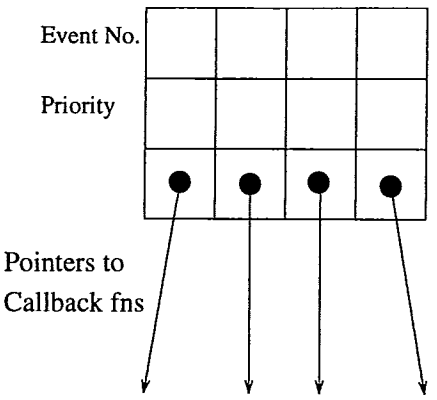


**Figure 1 : Structure of an individual in the MP population**

to by *Main_loop* is called repeatedly until interrupted (assuming they have a higher priority) by one of the other functions.

### Choosing Events

Although it would be possible, and indeed, more useful, to allow evolution to choose under what conditions an event should be registered, this paper will only consider the most simple model, which consists of a fixed set of events. In this case, every program consists of four registered events, and it is the responsibility of the event handler to monitor the robot's sensors and call the corresponding functions. For the purposes of this paper, the MP individuals contain information for four events which are called under the following circumstances.

- The robot is involved in a collision (Cld_event)
- The robot is struck by a missile (Hit_event)
- The robot sees another robot (See_event)
- No other event is active (Ord_event)

An MP individual contains a priority level for each of these events, as well as a pointer into the CB population to the individual to be used for that event. When a robot's sensor registers information, such as being struck by a missile, this information is retained for a number of time steps before being set to a blank value. An event can only interrupt another one if it has a higher priority, and if an event is interrupted its callback function will only be called again if the condition that originally caused it still holds.

The individuals in the MP population were arranged in a similar demetic structure to the individuals in the original experiments, and fights / reproduction were arranged in the same way. The individuals in the CB population were also put into a demetic structure, which made it a trivial task for the MP population to keep track of what individuals they were pointing at. Again, crossover and reproduction was implemented using a deme size of three.

Each fight was conducted as follows:

- Select MP robots to fight.
- Using CBs pointed to by MPs, resolve conflict.
- Award points to MPs in the same way as before.
- Each CB used by an MP gets the same reward. The rewards given are as follows:
  1. +3 for each CB.
  2. +1 for each CB.
  3. -1 for each CB.

It is possible for a CB individual to be pointed at by two or more MP individuals in the same fight, or even used for several events by the same MP individual, so points were awarded for *each time* a CB individual was pointed at.

## Early Results

The initial results for this simulation were quite disappointing, with the more successful members of the MP population using only one or two callback functions in a sensible manner, i.e. an individual fired if it saw another one, but did nothing if it was hit itself, or if involved in a collision. It was very rare that an individual was produced who made proper use of the event structure.

Another disappointment was the difficulty encountered when comparing the individuals - sometimes an individual who reacted appropriately upon seeing another robot performed better than an individual who took evasive action upon being shot, but other times it did not.

There were two reasons for these disappointing results. Firstly, the MP population became very stable very quickly, so newly created individuals in the CB population often didn't have an MP pointing at them, and so didn't even get tested. The second reason was the CB population trying to maintain a balance of radically different individuals - crossover between an individual successfully used as callback function for an MP being shot and an individual called when an MP didn't have any other active events often produced unviable individuals.

The solution to both of these problems was to divide up the CB population into several sub populations, one for each event.

## Multiple Callback Populations

This decision resulted in four CB populations. An MP individual now pointed at one individual in each population. Again, it was unusual for an MP individual to be produced that used the event structure properly, most using one or two good events, but pointing at less useful ones also.

One distinct advantage of this method, however, is that one could select good performing individuals from the CB populations in a similar manner to the original simulation - in this case there was no doubt which individuals were a good choice for each event. Like the original simulation, GP produced a number of different, but equally interesting individuals for each event. Individuals for the *See_event* for example, ranged from the aggressive

**(PN2 (REV (FIRE)) (FIRE))**
or
**(IF-Collide (REV 2) (FIRE))**

to the passive, who shied away from any contact with other robots

**(PN2 (REV 5) (RT 1))**

to the cautious, who only fired if it was absolutely necessary

**(IF-Collide[4] (FIRE) 1)**

Similarly, each event produced individuals who adopted interesting and different approaches. For the *Cld_event* individuals either moved away from the collision very quickly, turned to face the offending individual, or simply started firing. *Hit_event* produced individuals remarkably similar to *Cld_event*, but, as both these events cover somewhat similar circumstances this was not too surprising.

The final event is the *Ord_event* and these individuals fell into two categories noted earlier - Sloths and Spinners. When no other events were called, individuals either scanned the arena slowly, or simply waited for something to happen.

It should be noted that only the successful individuals are reported in this paper. Other individuals also appeared, who would warrant interest because of their unusual and often amusing tactics - individuals who tried to ram others for example, or those turned their back on individuals who shot them. Due to the pressures of space, however, this paper will only concern itself with viable and useful individuals.

The problems of skewed fitness measures still existed, however, with several individuals not even being tested. Again, this was due to no MP individuals pointing at them. Other individual had an over abundance of MP individuals pointing at them even though they themselves didn't do much to help that individual. These parasitic individuals tended to take over large portions of the population and often survived from Generation 0 until the end of the run. One possible solution to this is to calculate the fitness of the CB individuals based on their *average* performance over all the fights they are involved in, which would reduce an individual scoring highly by simply being involved in a lot of fights.

[4] Although it may appear strange that the callback function for the *See_event* investigates a sensor associated with another event, this often occured, particularly in the case of a high-priority event.

### Digesting the results

At this point we have a number of genuinely useful individuals for each event, but no clear indication as to how they should be combined, or what priority should be associated with each event. As mentioned above, the MP individuals from the runs that produced these individuals, although serving to provide a fitness measure, are not particularly useful now.

The solution to this problem is to run the simulation a second time, in a slightly modified form. This time, the CB populations are seeded with individuals produced in the first run, and no crossover takes place in those populations. This time so, only the MP population is evolving, and "interesting" individuals from these simulations would be candidates for the *best-of-run* individual, which can be selected in the same manner as before.

There was an enormous variety of individuals produced who were more than able to hold their own in the arena. Most individuals reacted in some way to each event, the most common being as follows :

- See_event : Fire or run away.
- Hit_event : Turn to face opponent, turn slowly or run away.
- Cld_event : Turn to face opponent, turn slowly or run away.
- Ord_event : Scan the arena, run around or do nothing.

The best individual to appear was very simple, but quite agressive, and, once it found an opponent would not stop firing until the opponent either got out of the way or died. This robot did not win every fight it was involved in, but in the tournament at the end of the run it performed the best. In the fights it did lose, it usually was because it backed into a wall, and, while being shot by some individual it couldn't see, kept trying to reverse.

**(Reg_event See_event 68 (FIRE) )**

**(Reg_event Cld_event 37 (RIGHT 5) )**

**(Reg_event Hit_event 23 (REV 1) )**

**(Reg_event Ord_event 1 (RIGHT 3) )**

A number of other individuals also appeared, using various combinations of the above strategies, with different priorities. Few of the successful robots did nothing when there were no events happening, and those that did were often struck several times before they reacted to an attack. The next best strategy was to shoot only when not under attack. Individuals adopting this strategy had a low priority for *See_event*, choosing instead to take evasive action if they came under attack.

## Best-of-Paper Results

The big question of course, is what is the best individual produced by any of the simulations? In the end it came down to a competition between those in Appendix A, the best three individuals that appeared in each simulation being chosen to fight in a similar demetic tournament to those used in the simulations. The best four individuals (three of whom were event-driven individuals) were then put into a round-robin tournament together, with the eventual winner being the individual described above.

Is this individual the best possible? Probably not. There is an enormous amount of tweaking that can be done, and a huge variety of other approaches to evolving an individual - from varying the deme size, to the fight size, to the kind of functions available to the robots - the list is endless.

## Discussion and Future Work

This paper presents a new benchmark problem for GP. Unlike many benchmarks, there is no known optimal solution, and, because the evolved controller program is abstracted away from the robot to be controlled, there are a huge number of ways to approach the problem. Every part of the problem, from the functions and terminals available, to the tournament size, to crossover implementations, can be changed without affecting the main problem - to produce interesting and viable behaviours - in any way.

Due to the nature of the problem, it is easy to compare two (or more) different solutions against each other, to test the performance of GP under varying conditions. It is even possible that sometime in the future there could be a GPRobot tournament, where individuals bred under different conditions could be tested against each other. Such is the particularly open nature of those in the GP community, it is hoped that if such a tournament does take place, that, unlike competitions in games such as those mentioned earlier, *all* information about a robot be made available, including code and the method used to produce the individual. In the event of such a tournament taking place, it is likely that the individual in the previous section can be bettered - and that is the whole point of this benchmark, that individuals from various approaches can be directly compared, or even watched on screen.

This paper also introduces the notion of teams of specialist individuals in GP and the evolution of event driven programs, where teams of individuals cooperate with each other in competition with other teams. This method produced the best individual found, and the area warrants further investigation, particularly in the areas of evolving the conditions under which an event is deemed to have occured, which would allow the system to become truly automatic. Potentially, event driven programming has an enormous number of applications area. This paper has served to show how programs requiring any number of events may be evolved.

This is very much a report on work in progress, and little effort was made to optimize any part of the sim-

ulations. It was clear that some degree of mutation is needed in the MP population to prevent new CB individuals from being ignored. Also, as noted above, calculating the CB individuals' fitness as average rather than a sum could well reduce any skewed measurements.

The event driven approach can even be used in applications that don't employ events, Appendix B shows some individuals rewritten in this way. The only difference in individuals rewritten is that they no longer employ pre-emption.

## Appendix A - Candidates for the Best-Of-Paper Individual

The following individuals were considered when choosing the Best-of-Paper individual.

From the initial simulations:

*Spinners :*

- **(IF-See (FIRE) (LEFT 2))**
- **(IF-See (FIRE) (LEFT (IF-Missile (bearing_of_enemy) 5)))**

*Panicker*

- **(IF-Hit (AHEA (RIGHT 5)) 1)**

*Event Driven Individuals :*

- **(Reg_event See_event 96**
  **(FIRE) )**
  **(Reg_event Cld_event 73**
  **(PN3 (LEFT 5) (RIGHT 5) (AHEA (RIGHT 5) ) )**
  **(Reg_event Hit_event 23**
  **(AHEA (LEFT (AHEA (LEFT 2) ) ) ) )**
  **(Reg_event Ord_event 1**
  **(LEFT 3) )**

- **(Reg_event See_event 68**
  **(FIRE) )**
  **(Reg_event Cld_event 37**
  **(RIGHT 5) )**
  **(Reg_event Hit_event 23**
  **(REV 1) )**
  **(Reg_event Ord_event 1**
  **(RIGHT 3) )**

- **(Reg_event See_event 34**
  **(PN3 (REV (FIRE) ) (FIRE) (REV 1) ) )**
  **(Reg_event Cld_event 23**
  **(RIGHT 3) )**
  **(Reg_event Hit_event 82**
  **(REV 1) )**
  **(Reg_event Ord_event 17**
  **(RIGHT 3) )**

## Appendix B - Event Driven individuals rewritten without events

The Event Driven individuals shown in Appendix A are rewritten below. Functionally, they are almost identical in either form, in that the priorities remain intact, but it is no longer possible for pre-emption to take place.

```
(IF-See (FIRE)
  (IF-Cld (PN3 (LEFT 5) (RIGHT 5) (AHEA
   (RIGHT 5) ) )
    (IF-Missile (AHEA (LEFT (AHEA (LEFT
    2) ) ) )
     (LEFT 3) ) ) )

(IF-See (FIRE)
  (IF-Cld (RIGHT 5)
    (IF-Missile (REV 1)
     (RIGHT 3) ) ) )

(IF-Missile (REV 1)
  (IF-Hit (PN3 (REV (FIRE)) (FIRE) (REV
   1))
    (IF-Cld (RIGHT 3)
     (RIGHT 3) ) ) )
```

## References


Smith, R.E. 1992 : Studies in Artificial Evolution. PhD Diss., Dept. of Computer Science, UCLA.

Dewdney, A. (1984) : In a game called core war hostile programs engage in a battle of bits. *Scientific American* 250:14-23.

D'haeseleer, P. and Bluming, J. : Effects of Locality in Individual and Population Evolution. In Advances in Genetic Programming. Ed. K. Kinnear Jr. Cambridge : MIT Press

Hillis, D. (1991) : Co-evolving parasites improve simulated evolution as an optimization procedure. In Artificial Life II Ed. C. G. Langton. Addison-Wesley.

Jannink, J. (1994) : Cracking and Co-evolving Randomizers. In Advances in Genetic Programming. Cambridge : MIT Press

Koza, J. (1992) : *Genetic Programming.* Cambridge : M.I.T. Press.

Timin, M. : *Robot Auto Racing Simulation* available through anonymous FTP from magdonaz.mcaffee.com

Reynolds, C. : Evolution of Obstacle Behaviour: Using Noise to Promote Robost Solutions. In Advances in Genetic Programming. Cambridge : MIT Press

Reynolds, C. : Competition, Coevolution and the Game of Tag. Forthcoming.

Rognlie, R. (1993) : *C++Robots* available through anonymous FTP from ftp.netcom.com.

Ryan, C. (1994) : Pygmies and Civil Servants. In Advances in Genetic Programming. Ed. K. Kinnear Jr. Cambridge : MIT Press

Ryan, C. (1994) : Racial Harmony in Genetic Algorithms. Forthcoming.

Ryan, C. (1995) : Racial Harmony and Function Optimization in Genetic Algorithms - The Races Genetic Algorithm. Forthcoming.

Schick, B. (1994) : *Robowars* available through anonymous FTP from psycfrnd.interaccess.com.

Siegel, E. (1994) : Competitively Evolving Decision Treees Against Fixed Traiing Cases for Natural Language Processing. In Advances in Genetic Programming. Cambridge : MIT Press