Virtual Witches and Warlocks: A Quidditch Simulator and Quidditch-Playing Teams Coevolved via Genetic Programming

Raphael Crawford-Marks, Lee Spector, and Jon Klein

Cognitive Science
Hampshire College
Amherst, MA 01002
{rpc01, lspector, jklein}@hampshire.edu

Abstract. Games make excellent challenge problems for Artificial Intelligence. Two-player turn-based games (Backgammon, Checkers, Chess) are easy to program, and AI players can be benchmarked against humans of varying skill levels. Recently, more complicated real-time team games have received attention because of their dynamic environments and the necessity for coordination. The RoboCup Soccer Simulator is the most popular and well-known of these environments. However, the soccer simulator is restricted to only two dimensions, and does not realistically model physics. In 2001, Spector et al. proposed creating a simulator of the imaginary game Quidditch from the Harry Potter Books by J.K. Rowling. This article describes such a simulator and the coevolved quidditch-playing teams created for it using Genetic Programming.

1 Introduction

Quidditch is the most popular game in the imaginary world of wizardry created by author J.K. Rowling in her series of Harry Potter books [RG98]. It's a fast paced game, something like a high-flying mix of basketball and soccer. Quidditch is more complex than most real sports, with fourteen players on two teams, four balls of three different types, six goal hoops, and, most significantly, flying broomsticks. Implementing quidditch as a challenge problem for AI, specifically automatically programmed AI, was proposed by Spector et al. at GECCO-2001 [SMR01]. This article describes a full-featured Quidditch Simulator implemented in a robust physical simulator called BREVE, and the teams of software agents that were evolved to play it.

Using games as a problem domain for Artificial Intelligence is not a new idea. Two-player strategy games like backgammon, checkers, and chess have all been extensively explored problem domains for various AI techniques [Tes92], [SLLB96], [SS92]. Computer Go is currently a very active area of research [BC01]. In the last twenty years, agent-based AI, known as Distributed Artificial Intelligence (DAI) and Multi-Agent Systems (MAS), has grown in popularity, and researchers have sought out games that better fit the new paradigm. Robot soccer has emerged as one of the most popular game-based problem domains for agent-based AI.

Since the publication of the Robot World Cup Initiative [KAK$^+$97], robot soccer has emerged as a challenging and entertaining environment to test the abilities of software agents, and evaluate the strategies we use to program them. However, the RoboCup simulator has some significant drawbacks. It doesn't model any physics save for collisions, and even those are modeled very simply. RoboCup soccer also ignores the third dimension, and in effect simulates a collective air hockey game, not soccer. Virtual quidditch is an ideal candidate to succeed virtual soccer as the next step in complex, dynamic challenge environments for AI research.

Creating a Quidditch Simulator presented some unique challenges. Being an imaginary game, the rules weren't completely described, and some of the imagined rules didn't really make sense once the game was implemented. However, with a few modifications to Rowling's vision, simulated quidditch is as fast-paced and exciting as she described. Evolving quidditch-playing programs also presented several challenges, such as designing an appropriate fitness function and trying to tweak parameters to get the best evolutionary performance for a relatively small number of fitness evaluations. This article describes the Quidditch Simulator, the process by which quidditch-playing teams were evolved, and the results of the evolutionary runs.

2 Quidditch Simulator

The Quidditch Simulator (QS) is written in a software package called BREVE. The QS implements the rules of quidditch as described in [WR01], [RG98] as faithfully as possible, though some changes were made to improve playability or reduce programming overhead. These changes are detailed in [CM04]. The latest source code of the quidditch simulator is available at the author's website.[1]

2.1 BREVE

BREVE is designed for the rapid simulation of decentralized systems and artificial life [Kle02]. It efficiently handles the often difficult and time-consuming tasks of physics simulation and 3D rendering in real time. Its object-oriented scripting language, steve, is powerful yet easy to use, and allows the programmer to take full advantage of BREVE's built-in class hierarchy. Detailed documentation on BREVE as well as the latest BREVE downloads can be found at http://www.spiderland.org/breve.

[1] http://hamp.hampshire.edu/~rpc01/vww.html

2.2 The Game of quidditch

Balls

The Quaffle The Quaffle is a round, inflated leather ball, painted red. In the early 18th century, witch Daisy Pennifold enchanted the Quaffle to fall as though sinking through water. The "Pennifold Quaffle" is still used today. In 1875, another enchantment was added to the Quaffle: a "gripping charm" allowing Chaser to easily keep hold of the Quaffle with one hand.

The Bludgers Bludgers started out as enchanted rocks, sometimes carved into the shape of a ball. Modern Bludgers are heavy iron balls, 10 inches in diameter. Bludgers are bewitched to chase the player closest to them. Therefore Beaters must try to knock Bludgers as far away from their teammates as possible.

The Golden Snitch The Golden Snitch is a walnut-sized golden ball with thin, translucent wings. It is fast, highly maneuverable and semi-intelligent, employing all its abilities to avoid being caught by either Seeker.

Players

The Keeper Keepers are like soccer goalies. They hover near their own goals to fend off shots from opposing Chasers. Keepers have all the same abilities as Chasers, so they do not exist as a distinct player class in the Quidditch Simulator.

The Chasers Chasers are somewhat analogous to soccer forwards. They can hold on to the Quaffle, pass it back and forth, and throw it at the opposing team's goal hoops. Normally there are only three Chasers per team, but in the Quidditch Simulator there are four because there is no Keeper.

The Beaters Beaters defend their teammates from the Bludgers. They are equipped with wooden bats to knock Bludgers away from their teammates.

The Seeker The Seeker is tasked with capturing the Golden Snitch. He or she is usually the fastest and most agile player on the team. When the Golden Snitch is caught, the capturing team is awarded one-hundred and fifty points, and the game ends.

Gameplay Quidditch is played over an oval-shaped field five-hundred feet long and one-hundred and eighty feed wide. This is called the Pitch. At each end of the pitch are three goal hoops. *Quidditch Through The Ages* does not specify the height of the goal hoops. In the Quidditch Simulator, they are 15 meters high.

At the start of the game, all players are grounded on their team's half of the pitch. The referee whistles, and the balls are released at midfield. The Quaffle is thrown into the air by the referee. At this point the quidditch match has begun.

In a typical (imagined) game, as soon as the referee whistles the start of the game the Keepers rush to their respective scoring areas to defend the goals (there are no Keepers in the Quidditch Simulator, if this behavior is adaptive then hopefully it will be adopted by one of the four Chasers). The Chasers lift off and scramble after the Quaffle. The Beaters track the Bludgers and assume strategic positions to defend their teammates. The Seekers will often climb high into the air to get a good view of the whole pitch. They circle around the pitch until spotting the Snitch, at which point they go into high-speed pursuit. The game ends when one of the Seekers catches the Snitch, or by the mutual consent of both team captains. According to the J.K. Rowling's books, Quidditch games have lasted anywhere from 5 minutes to several weeks. In the Quidditch Simulator, games are held to a strict time limit, starting at 10 seconds and increasing to several minutes as the run progresses.

2.3 The Simulator Architecture

BREVE provides a robust class hierarchy upon which simulations can be built using an object-oriented language called steve. The Quidditch Simulator makes heavy use of inheritance to implement the various actors in the simulation. Detailed documentation of the implementation of the simulator is beyond the scope of this paper, but details are available in [CM04].

The simulation mechanics are managed by the BREVE engine, which calculates velocities (taking friction and other forces into consideration), resolves collisions and manages other simulation-wide information. Players and balls are able to move by applying forces to their center of gravity in any direction, as if equipped with a single thruster that could be instantly pointed in any direction. Sensing is achieved with a noisy omnidirectional radar. Players can sense the location of any game object, but the sensed location is subject to noise, the amount of which is proportional to the players' distance from the object. No facilities for agent communication are currently implemented, but these are on our short-term development agenda.

3 Coevolving Quidditch-Playing Teams via Genetic Programming

Quidditch-playing teams were coevolved in the Quidditch Simulator. Since three of the four balls are semi-intelligent, a separate population of balls was also coevolved along with players. Each genome consisted of several program trees, one for each member of the team (see Figure 1). Programs were expressed in the Push programming language.

3.1 Push

Push is a stack-based programming language developed by Spector with an eye towards automatic programming, specifically genetic programming. Push programs look very LISP-like, with symbols grouped in balanced (and possibly nested) pairs of parentheses. However, Push programs are interpreted very differently from LISP, and in this respect are more similar to programs in other stack based languages like Forth or Postscript. For a detailed description of the Push programming language, see [SR02], [SPK03].

3.2 Evolutionary Process

A separate BREVE simulation called Quidditch Evolver (QE), manages the evolution of quidditch teams. After generating random initial populations of balls and players, the QE places teams in a game pool. Pools are filled by selecting individuals in sequence from the populations and then selecting n teams at random. For a player poplation {p1, p2, p3, p4} and ball population {b1, b2, b3, b4}, the first game pool would have {p1, b1} and then two randomly selected player-teams, say {p2, p3}, to form the game pool {p1, p2, p3, b1} (the next game pool would have {p2, b2} and two randomly selected teams). The teams then play each other in a round robin (every team plays every other team). For example, the pool we just described would be played out in 3 games: { {p1, p2, b1}, {p1, p3, b1}, {p2, p3, b1} }. Ball-teams never compete directly with one another, only against player-teams. At the end of each game, the QE gathers fitness data from an output file left by the Quidditch Simulator.

Once evaluation has been performed, the QE probabilistically selects individuals based on fitness to populate the next generation. There are four genetic operators that facilitate this: crossover, mutation, trading, and reproduction (details available in [CM04]).

3.3 Fitness Scoring

Fitness scoring is very similar to the system of "lexical dominance" used by Andre and Teller [AT99]. This type of fitness scoring gives tiny rewards for simple behaviors that could lead to better complex behaviors, like moving around and throwing the Quaffle. A very large fitness reward is given for scoring (through goals and Snitch-catching). This large fitness reward is one or more orders of magnitude larger than the small fitness "nudges" and effectively trumps them once teams learn to score. A detailed description of the fitness function for players and ball can be found in [CM04].

4 Results

Several evolutionary runs were performed. Results are summarized below. Runs were performed on a 13-node cluster of Linux machines with dual Intel Xeon 1.8GHz processors. Even with such significant computing power, the latest and longest run (which evaluated only 26,260 player and ball-teams) took well over 48 hours to complete. The results described below are from this latest run, which ran for 101 generations with a population size of 20 player-teams and

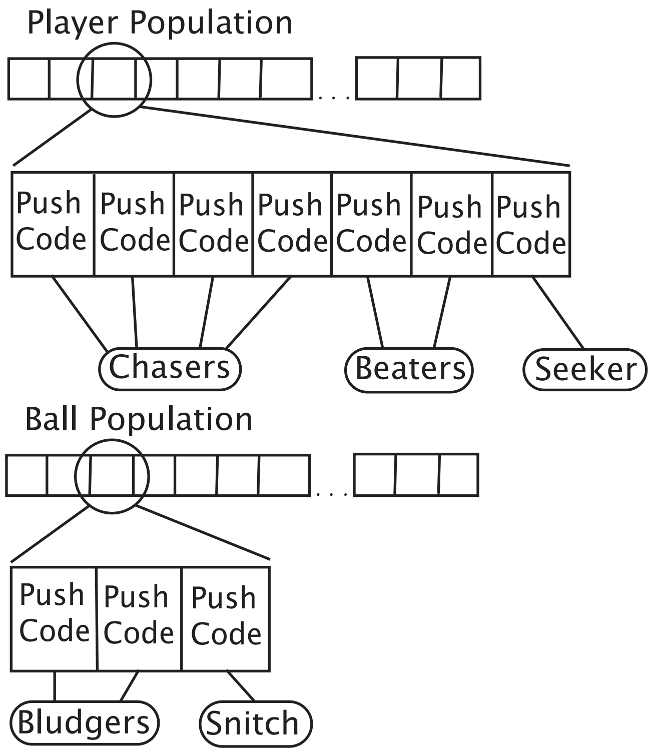

Fig. 1. Each individual in the population is an entire PushTeam.

20 ball-teams per node, for a total population size of 260 player-teams and 260 ball-teams.

Evolved programs contained instructions of two types: instructions that performed computations and modified the stacks, and instructions that were actually callback functions to the BREVE simulation. These callback function would either return sensor information or cause the agent to perform an action in the simulated world.

Ball-Teams Initial randomly-generated ball-teams performed very poorly and some displayed behaviors that were the exact opposite of what was desired. Some Snitches would fly right at Seekers, instantly ending the game. Some Bludgers would chase after Beaters getting repeatedly whacked all over the pitch. Luckily, these behaviors did not last very long.

By Generation 10, all three balls on the best teams have evolved halfway decent, if very simple, behaviors. Those behaviors run along the lines of mindless pursuit and mindless flight. The two Bludgers will simply accelerate directly at

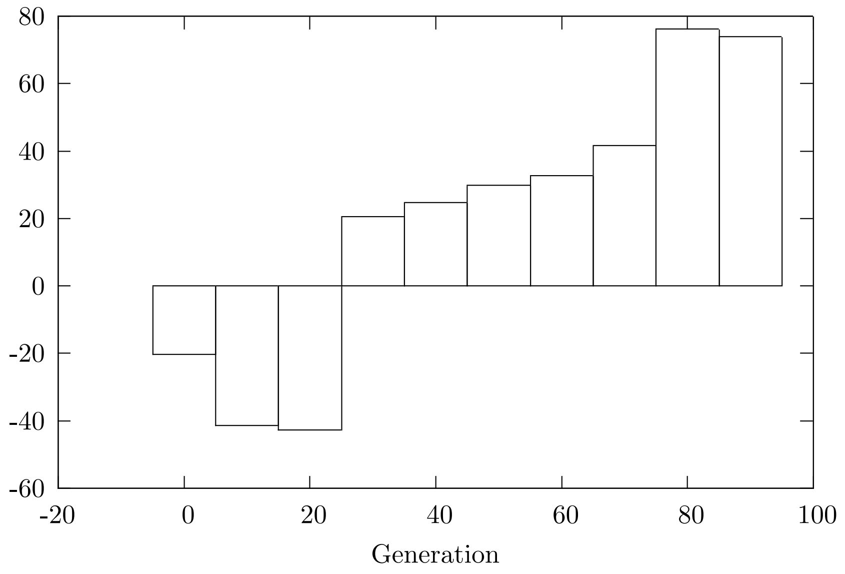

Fig. 2. Average Relative Fitness in the Generation 90 Tournament of Champions. This shows the average fitness score (as determed by the fitness function described in section 3.3) for each team after playing in a round robin pool composed of the best player-teams of every tenth generation.

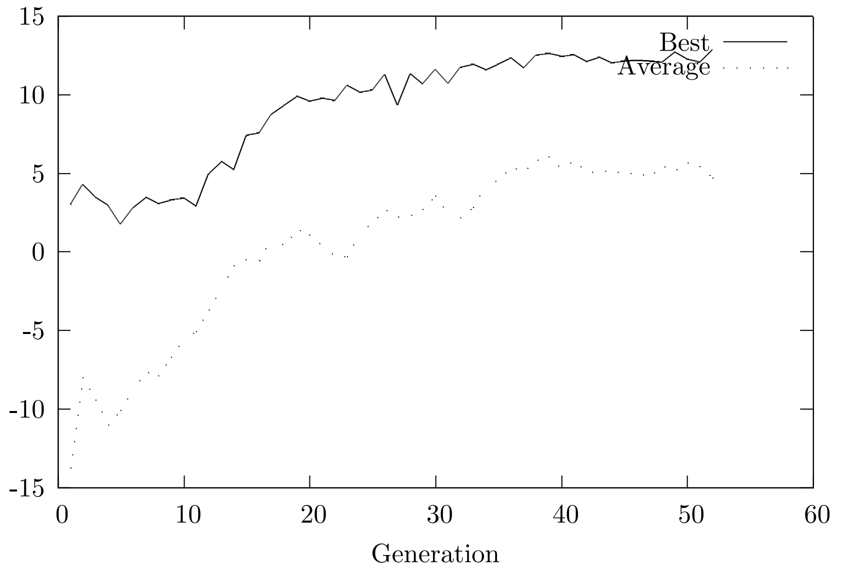

Fig. 3. Average and best fitness of ball-teams across all nodes.

the nearest (non-Beater) player, while the Snitch accelerates directly away from the nearest Seeker. While the relentless pursuit of Bludgers is quite effective, relentless flight turns out to be somewhat problematic for the Snitch. The Snitch often only flees the nearest Seeker, sometimes flying directly towards the other Seeker and not realizing it until too late. Other times, the Snitch will flee and flee and flee in one direction until it bumps up against the "invisible bubble" border of the world, and is bounced right in to one of the Seekers pursuing it.

In later generations, Bludgers appear to continue with the relentless pursuit strategy. Since they are scored based on the number of collisions with non-Beater players, relentless pursuit turns out to be a fairly effective strategy. The Snitch, on the other hand, evolves a more complex avoidance algorithm that takes in to account the locations of both Seekers, and doesn't simply flee mindlessly.

The average fitness of ball-teams for every generation is a good indication of how often Snitches are being caught. Ball-teams receive a fitness hit of 20 points if the Snitch is caught, which in practice will always result in a negative overall fitness score. Therefore, negative average fitness scores can be evidence of frequent Snitch-catchings. Figure 3 shows the overall average ball-team fitness score across the entire run. As you can see, by Generation 25, average fitness scores go positive and never turn back.

Player-Teams The initial random teams performed very poorly. Many players didn't move, and those that did often wandered randomly or chased their teammates. However, the code stack of each player was seeded with `( move )` and `( throw )`, and the vector stack was seeded with `my-object-of-interest`. Therefore, simply executing `move`, `move-as-fast-as-possible`, or `CODE.DO` would cause a player to pursue its object of interest (Chasers would pursue the Quaffle, Beaters the nearest Bludger, Seekers the Snitch). For Chasers, if `throw` or a second `CODE.DO` appeared in their code, they would pursue the Quaffle and throw it at the nearest opponent goal. Indeed, some of the best of Generation 0 programs contain one or two players that do this. Another adaptive behavior exhibited by some of the best teams from Generation 0 is a Seeker that relentlessly pursues the Snitch. Most Generation 0 Snitches have poor avoidance behavior, if any, and are easily caught by simply charging it.

By Generation 10, most of the good teams have one or two Chasers who perform the "chase and throw" behavior, and a Seeker that actively pursues the Snitch. Occasionally, a Beater will pursue Bludgers as well. The ball-teams have improved significantly by Generation 10, and the best ones will have a Snitch that is hard to catch, and two Bludgers that aggressively attack the nearest players.

Around Generation 20 or so, ball-teams have evolved good behaviors. Both Bludgers attack nearby players, while the Snitch is almost uncatchable. The latter forces player-teams to score goals in order to win, since they cannot rely on their Seekers. Here is where "kiddie-quidditch" really begins to emerge. The name is inspired by Luke et al.'s observation of "kiddie-soccer" behaviors in [LHF+97]. Essentially, the evolved Chasers play quidditch much like six-yearolds play soccer. They all charge the ball and throw it towards the goal, with no regard for defense. As in [LHF$^+$97], kiddie-quidditch is a very attractive suboptimal behavior, discovered fairly quickly by every evolutionary run. Many of the adjustments to various system settings from run to run were done in an attempt to help the population escape from kiddie-quidditch into more fertile areas of the search space.

No defensive behaviors or new offensive strategies emerged from generation 30 - 90, save one slight modification to the kiddie-quidditch strategy. Around Generation 30, one of the four Chasers would hold on to the ball and carry it to the opposing goal. There, it would orbit the goal until the ten second time limit on possession expired and it was forced to drop the Quaffle. The other three Chasers would still simply throw the ball towards the goal. This appears to be advantageous because it brings the Quaffle very close to the opposing team goals. Because of sensor noise, long-range shots are often wildly innaccurate. When the Quaffle is dropped near the goal by the ball-holder, it is so close that any shot at the goal is almost always 100% accurate.

The best end-of-run teams still played kiddie-quidditch, but many of the teams showed the beginnings of separate offensive and defensive strategies. On some teams, one or two Chasers would hang out in front of their own goals while the rest of the Chasers continued to play chase-and-throw kiddie-quidditch. Occasionally, these defensive Chasers would intercept shots by the opposing team. One team appeared to go a step further: in addition to a defensive Chaser, an offensive Chaser would fly around near the opposing teams goals, and then pursue the quaffle when it came near.

Results from the tournament of champions shows a steady increase in player-team performance until the last twenty generations, when relative fitness scores level off. This is evidence that kiddie-quidditch is a strong enough local optimum to halt evolutionary improvement in its tracks. However, the goal-guarding behavior observed in the last few generations may be the beginnings of defensive strategies and the escape from kiddie-quidditch.

Perhaps with some more time, evolution could escape from kiddie-quidditch altogether and move on to better strategies. Many genetic programming runs evaluate fifty thousand, a hundred thousand or more individuals before finding a solution. The 26,260 individuals evaluated is a relatively small number, especially given the combinatorics of the function and terminal set. More time, a larger population size and more computational muscle could yield significantly better quidditch-playing teams.

5 Conclusions and Future Work

5.1 Quidditch Simulator

This work is a promising first step in establishing virtual quidditch as a rich and challenging problem for AI. The rapid run time, accurate simulation of newtonian mechanics and 3D world of the Quidditch Simulator are significant improvements over the RoboCup Soccer Simulator. However, there are several areas in which the simulator still does not match up with the RoboCup simulator:

Stamina Players (and perhaps balls as well) should have a limited amount of stamina that is used up by acting in the game world and is regenerated at a constant rate. The more effort players put into an action (i.e. the faster they move or the harder they throw) the more stamina is consumed. Players cannot expend more effort than they have stamina.

Communication Players should be allowed to communicate by "talking" to one another. This type of communication is implemented in the RoboCup Soccer Simulator with `say` and `hear` commands. A similar implementation would work well in virtual quidditch.

Vision-like sensors Currently, agents have noisy but still very unrealistic sensors. They are equipped with omnidirectional radar that degrades in accuracy over large distances. A more vision-like sensor system would be much more realistic. There are different ways to implement such a feature, depending on how vision-like one wants the sensors to be. One easy way of implementing such sensors would use the noisy radar that agents currently have, but limit its scope to a forward-looking cone.

Client/Server architecture Using network code currently being developed for BREVE, the QS should be altered to run as a server, with player and ball-teams connecting remotely. This would enable long-distance competitions and create the possibility of pairing evolution with the computational muscle of distributed computing.

5.2 Evolving Quidditch Teams

On the evolutionary front, this first attempt at evolving quidditch-playing teams was not as successful as was hoped for. However, the pitfalls encountered have shed light on potential refinements to future experiments. Most significantly, the results showed how distinct the different player roles are, almost to the point that quidditch can be thought of not as one game, but three games being played simultaneously on one field. This is not entirely true, because all three subgames factor in to the overall result, and each player wins or loses along with the team, not based on individual performance. A good analogy for this is american football, where players assume specific roles on either offense, defense or special teams that are very different from one another.

There are several refinements that could help evolution escape the from kiddie-quidditch:

Separate throw and move trees In [LHF+97], soccer-playing softbots had two program trees, one for movement and one for ball-kicking. Given the apparent difficulty of evolving control structures to decide when to throw the ball and when to hold on to it, giving Chasers a separate program tree dedicated to ball throwing may help evolution overcome that difficulty. The move tree would be executed at every timestep, and the throw tree would be executed if the agent has possession of the Quaffle.

Separate populations for each player and ball class Quidditch can be thought of as three sub-games being played in parallel. As such, it may be more effective to separate the different classes of players and balls into separate evolving populations, and create a customized fitness function for each.

Shared ADFs for entire teams Andre and Teller successfully evolved team-wide Automatically Defined Functions (ADFs) in [AT99]. ADFs have been shown to significantly increase problem-solving power and program parsimony in genetic programming runs [Koz94]. Push has the capability to evolve modular programs, but any evolved modularity in player program trees would be useful only to that specific player. Giving each team (or each group of players of a specific class) a set of evolvable ADFs could help teams evolve building-block behaviors that can be combined to create more complex offensive and defensive strategies.

Internal Memory Another big obstacle to evolving more complex behaviors is the lack of any capability for storing information across over time. Internal memory would give agents the ability to use internal state when deciding what to do, and could also enable them to make calculations about the trajectory of other objects in the game world by comparing previous and current locations. Internal memory could be easily implemented by not clearing the Push interpreter stacks after each timestep, or via persistent variables implemented using Push's NAME data type.

Acknowledgements

This effort was supported by an NSF Director's Award for Distinguished Teaching Scholars (to Spector), by NSF grant EIA-0216344, and by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30502-00-2-0611.

References

[AT99] D. Andre and A. Teller. Evolving team darwin united, 1999.

[BC01] Bruno Bouzy and Tristan Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[CM04] Raphael Crawford-Marks. Virtual witches and warlocks: Computational evolution of teamwork and strategy in a dynamic, heterogeneous and noisy 3d environment. Division III (senior) Thesis, Hampshire College, 2004. http://hamp.hampshire.edu/~rpc01/div3.pdf.

[KAK$^+$97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997. ACM Press.

[Kle02] Jon Klein. Breve: a 3d environment for the simulation of decentralized systems and artificial life. In *Artificial Life VIII: Proc. of the 8th Int. Workshop on the Synthesis and Simulation of Living Systems.* MIT Press, 2002.

[Koz94] John R. Koza. *Genetic programming II: automatic discovery of reusable programs.* MIT Press, 1994.

[LHF$^+$97] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. In *Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

[RG98] J.K. Rowling and M. Grandpre. *Harry Potter and the Sorcerer's Stone.* Scholastic, Inc., 1998.

[SLLB96] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK: The world man-machine checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[SMR01] Lee Spector, Ryan Moore, and Alan Robinson. Virtual quidditch: A challenge problem for automatically programmed software agents. In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 384–389, San Francisco, California, USA, 2001.

[SPK03] Lee Spector, Chris Perry, and Jon Klein. Push 2.0 programming language description. http://hampshire.edu/lspector/push2-description.html, 2003.

[SR02] Lee Spector and Alan Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, 2002.

[SS92] H. Simon and J. Schaeffer. The game of chess, 1992.

[Tes92] Gerald Tesauro. Practical issues in temporal difference learning. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 259–266. Morgan Kaufmann Publishers, Inc., 1992.

[WR01] K. Whisp and J.K. Rowling. *Quidditch Through the Ages.* Scholastic Press, New York, NY, 2001.