On the Genetic Evolution of a Perfect Tic-Tac-Toe Strategy

Gregor Hochmuth
Stanford University
Stanford, CA 94305
grex@cs.stanford.edu

ABSTRACT

This paper demonstrates how a genetic algorithm can be used to evolve a perfect tic-tac-toe strategy, which never loses a game it plays.

Introduction

Tic-Tac-Toe is an ancient game whose first traces date back as early as 1300 BC in Egypt. Also known as *"Noughts and Crosses"*, it is a classic match between two players, who alternate in marking spaces in a 3x3 grid, trying to put three of their own marks ("X" or "O") in a horizontal, vertical or diagonal row.

Although considered a simplistic pastime, it proposes an interesting problem to consider with regard to an optimal strategy. Given that a match is played with relatively few moves, optimality in terms of Tic-Tac-Toe takes a more defensive meaning compared to other games. Namely, an optimal strategy is only guaranteed *to avoid a player's loss*, whereas it cannot guarantee a win against any opponent. On the other hand, the smaller number of distinct game situations also suggests that such strategy could be potentially found and encoded using computational search methods, which motivated the project described by this paper.

While smaller in scale, Tic-Tac-Toe strategies still present an enormous search space, and although their set of possible game situations can be significantly reduced, it remains at impractical proportions for an exhaustive exploration. The challenge was therefore to devise an effective local search method for this problem, and based on their noted success given many similar problems, genetic algorithms were selected for this experiment.

As part of the larger family of evolutionary search methods, genetic algorithms combine problem-specific encoding and methods similar to natural selection in order to *evolve* a solution. In general, an implementation of such algorithm maintains a pool of individuals (the population), each of which carries an encoding that translates into some answer given the search problem, such as the complete specification of all parameters for building the most performing and least expensive satellite antenna. The better the answer that it carries, the higher the fitness an individual has. With each search step, a new population is created from the old where the fittest individuals are selected to exchange their specifications and reproduce. Over time then, beneficial features are expected to emerge from the population pool and to potentially produce an individual with an optimal answer. The reader is referred to Koza (1992) for a thorough introduction to genetic algorithms as the following discussion assumes familiarity with the domain.

The Genetic Haystack: Exploration of the Search Space

A genetic algorithm (GA) is most effective in situations, for which a well-defined problem offers a compact encoding of all necessary solution parameters. If this encoding grows too large and complex, the algorithm faces similar limitations of other local search methods and cannot be expected to find a global optimum. In considering the Tic-Tac-Toe strategy problem, it is at first important to find a suitable representation and to ensure that a GA can be effectively applied.

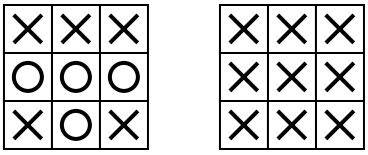
Tic-Tac-Toe Games and Situations

In general, a Tic-Tac-Toe strategy is most easily represented as a mapping for all possible game situations to a corresponding move that should be taken such as to (at least) avoid a loss. The size of the strategy space is therefore defined by the number of *all possible game situations*, which follows from to the question of how many distinct matches can be played.

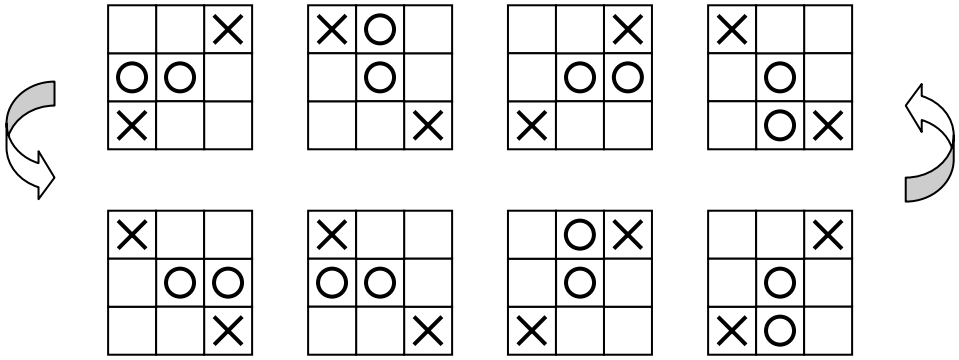
While it may seem unproductive to play Tic-Tac-Toe continuously, much time can in fact be spent by *counting* Tic-Tac-Toe. The game presents a 3x3 grid, in which each player can choose to put a mark; the number of unique

configurations, in which *all* 9 spots are marked is therefore *9!* (362,880), which gives the upper bound for the number of possible Tic-Tac-Toe *games* in terms of their final grids. The total number of all possible game situations, i.e. intermediate game stages, is given by 3^9 (19,683), representing one of three assignments ("X", "O", Empty), to all 9 positions (Schaefer, 2002).

However, many of these configurations are not plausible or even invalid; consider the following examples, in which the left game would have ended much earlier and the right state is entirely invalid:



In addition, the set shrinks further when realizing the inherent symmetry of the grid. Every game situation has 7 identical counterparts which can be obtained merely by flipping or rotating the board:



Both of these factors combined lead to a drastic reduction in the number of unique situations which a strategist can encounter. While there are a total of 23,129 possible Tic-Tac-Toe games to be played in long train rides, the number of unique situations is in fact only 827. This number was found empirically by exhaustively constructing all possible situations (3^9) and reducing the valid ones to their base cases as described above (Schaefer, 2002).

Defining Individuals and their Strategies

Given this manageable number of situations, it is therefore possible to define a representation that suitably encodes any Tic-Tac-Toe playing strategy in the context of a genetic algorithm. Specifically, an individual for this population maintains a mapping for each of the 827 situations to one of the legal moves, which can be taken in that situation. Depending on the remaining number of empty squares in a situation, the individual has a choice between at most 9 possible moves (empty grid) and 1 (last move); the strategy search space, the number of possibilities p_s for all s situations, is

$$\prod_s p_s = 2480$$

as determined through programmatic enumeration.

Strategy Encoding

Whenever an individual is asked to respond to a situation, the following process extracts the move encoded by its strategy mapping. Consider the situation:

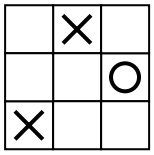


Figure 1. Example Tic-Tac-Toe Situation

First, all 7 symmetric situation are produced by applying transformations; for each of the 8 identical situations, a decimal equivalent is computed by means of a unique hash.

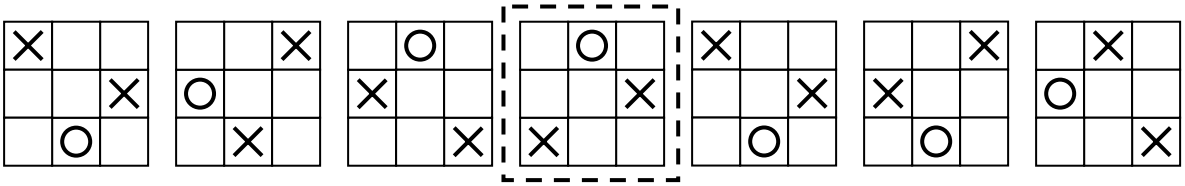


Figure 2. All situations identical to the example given in Figure 1, showing chosen *base case*

The situation with the lowest equivalent is considered the base case situation for its group (thus, there are 827 such base cases in the problem space). The base case then corresponds to an entry within the individual's mapping table; it returns a number *M* between 1 and *n*, where n is the highest index to represent a possible move in the situation:

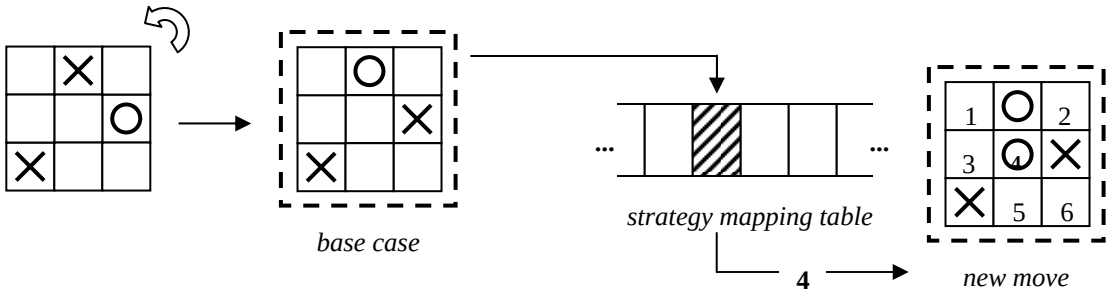
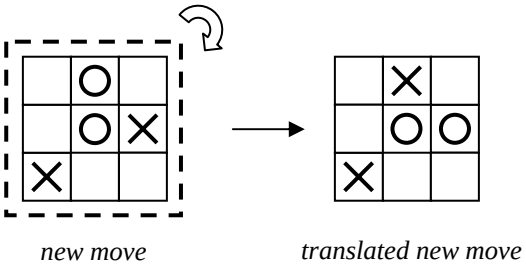


Figure 3. Strategy extraction from base case encoding

Applying the inverse of the transformation that was used to produce the base case, M is translated back into the context of the *original* situation in order to make the move:



Genome, Gene, Allele

In terms of the genetic algorithm, an individual's mapping table corresponds to its *genome* with 827 distinct *genes* to represent each situation as defined above. Lastly, an *allele* – the value of each gene – is defined by the move to be taken in that situation.

Performance Considerations

Given the several computations involved in extracting moves from the compact strategy encoding above, the final implementation of the algorithm made use of pre-computed look-up tables. One such table was initialized with all 3^9 possible game situations (including implausible ones); it was then used to quickly each situation to its unique *base case* and the corresponding transformation. A second table provided direct mapping from a base case situation to the index of the gene, which encodes an individual's response move. These enhancements provided significant time improvements during trial runs, especially with large population sizes, which require many exhaustive matches and strategy look-ups.

Genetic Operations

Having defined the genome of an individual, the next step in applying the genetic algorithm is to specify its operators and evaluation procedures. Based on its performance, each individual is assigned a fitness measure; the higher its measure, the more likely it is that an individual will participate in reproduction and pass on its features to the next generation. Once the parent generation is determined, the construction of offspring occurs by means of three operators: basic replication or mutation of a single individual, and crossover between two of them.

Strategy Fitness Evaluation

In the context of this Tic-Tac-Toe implementation, the fitness measure was computed by letting each chromosome play an exhaustive set of all possible matches and measuring its performance in terms of wins, losses, and ties.

The first match is started by letting the chromosome choose the first move on an empty board. Each possible move which an opponent could then take in response is added to a queue. For each game state de-queued from this list, the chromosome will choose its next move, and the set of all response states is added to the queue again. This iteration repeats until the queue is empty and all matches have resulted in a final game state: win, tie, loss.

An important second round of this process is played, in which the chromosome starts second; the initial queue therefore contains all possible opponent moves for an empty board. Starting as the second player is a serious disadvantage in Tic-Tac-Toe as it is rarely possible to win or even avoid defeat.

At the end of the two rounds, individual fitness is determined by the percentage of games not lost:

$$f(x_i) = \frac{n_{total}[x_i] - n_{lost}[x_i]}{n_{total}[x_i]}$$

Figure 4. Final Fitness measure

Initially, a *different* fitness measure had been used during the first trials of the experiment. It incremented an individual's fitness based on the outcomes of each match, using the point scale below:

Table 1 Point Scale used to determine fitness measure in the initial experiment set-up

	Player made first move	Opponent made first move
Game Won	+10	+15
Game Tied	+5	+10
Game Lost	+0	+0

Point assignments followed from the assumption that it is more difficult to win or tie a game when the opponent takes opening first move in the match. Furthermore, wins were valued higer than ties. Despite the rationale behind it, runs using this metric failed to produce optimal individuals and repeatedly suffered from local maxima stagnation (see *Results and Observations* below). In response, the evaluation measure was revised to the version presented before, and immediately resulted in the success of the first updated run.

In comparing the first and second method, it can be seen that the second prioritized individuals with fewer losses, which became crucial in the last stages of a run and was not addressed by original measure. Negative point values could have been used to penalize for losses with the first method, but the ratio, in which wins, losses, and ties would have required additional experimentation, whereas the second method is much simpler *and* more effective.

Replication, Mutation, and Crossover

Replication and mutation were included as two basic operators to ensure general variety in the reproductive process over time. The first merely copies one individual with probability $p_{replication}$ into the next generation without modification. Mutation can occur at each gene of an individual with probability p_{mutate}, by a new random allele value is chosen to replace the current one; p_{mutate} is usually very low since it is evaluated for each gene independently.

Because of the large genome in this problem, the crossover operator was modified slightly from its traditional implementation, in which a *constant* number of cross sites is chosen. In the implementation of this problem, the number of cross sites is *variable*: a probability $p_{crossover}$ is evaluated at each gene; genes are copied from *Parent A* until $p_{crossover}$ occurs, at which point, genes continue to be copied from *Parent B*. The next time $p_{crossover}$ occurs, the process flips back to *Parent A*, and so on. An example of this operation is given in Figure 5:

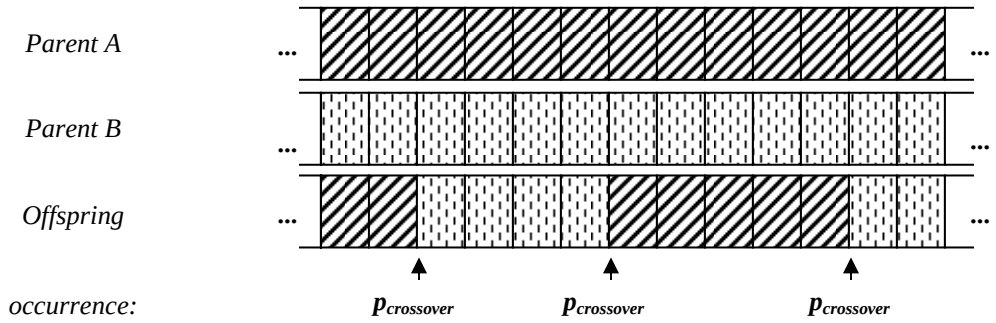


Figure 5. Illustration of redefined crossover operation

By performing crossover in this manner, the problem search space is rendered more accessible. For traditional crossover using one or two cross sites, the schemata for an 827-length genome are too specific to allow for effective gene exchange among parents and evolution in their offspring. Using a probabilistic approach, more than two cross sites are possible, which is desirable given the long gene sequence. Meanwhile, this difference manifests itself also in the different value given to $p_{crossover}$, which for many problems in genetic algorithms is around 0.90. In this implementation, due to its evaluation at each gene, the value is much lower, near 0.10.

Summary Tableau

To summarize then, these are the preparatory steps for applying the genetic algorithm to this specific problem:

Table 2 Preparatory Steps of the Tic-Tac-Toe Genetic Algorithm

Objective:	Find a perfect Tic-Tac-Toe strategy
Structure:	Fixed-length string of 827 characters to address all possible game situations; Alphabet size is 9 (index of the move to be taken); Mapping occurs from game situation to move taken.
Fitness cases:	Only one: an individual that loses no match
Raw Fitness:	Fraction of games not lost
Operators defined:	Replication, Mutation, variable Crossover
Population Initialization	Populations were always initialized randomly, no seeding heuristics were applied

Implementation

The code for the implementation of this problem relied on a custom-modified version of GENESIS, a C-based programming framework for genetic algorithms (Grefenstette, 1994). Specifically, modifications were made to accommodate for the specific data structure of the genome and for the particular refinement of the genetic crossover and mutation operators. In addition, the fitness evaluation procedure was re-written to perform the exhaustive match playing described earlier, see *Strategy Fitness Evaluation*.

Individual trials were performed on various Sun Blade 1000 machines, running Solaris 8, with 1GB of RAM, which were openly available as part of the UNIX cluster machines at Stanford University.

Results and Observations

Two iterations of trials needed to be performed before a successful run was recorded. Trials in the first set failed to find optimal individuals and continually remained at local maxima. The best individuals they produced were close to a perfect strategy but continued to lose small fractions of their matches (on the average 3 to 5 percent). This stagnation occurred when using the first version of the fitness measure, which evaluated individuals based on point values assigned to different match outcomes (see *Strategy Fitness Evaluation*). However, this measure failed to give priority to individuals who lost fewer games than others.

The measure was revised and dismissed the point system. With the updated version, an individual's fitness was given by the fraction of games *not lost* and using this improvement, the second iteration of trials surprisingly succeeded on the first run.

Fitness #1

Initially, the failures of the first iteration were attributed to an insufficient population size and several experiments with different values were performed. Interestingly, this did not correct the problem of local maxima, at which individuals were close to a perfect strategy. *Regardless* of the population size, 50 or 10,000, all eventually stagnated around the same low values for the fraction of games lost but never managed to escape it (Figure 6).

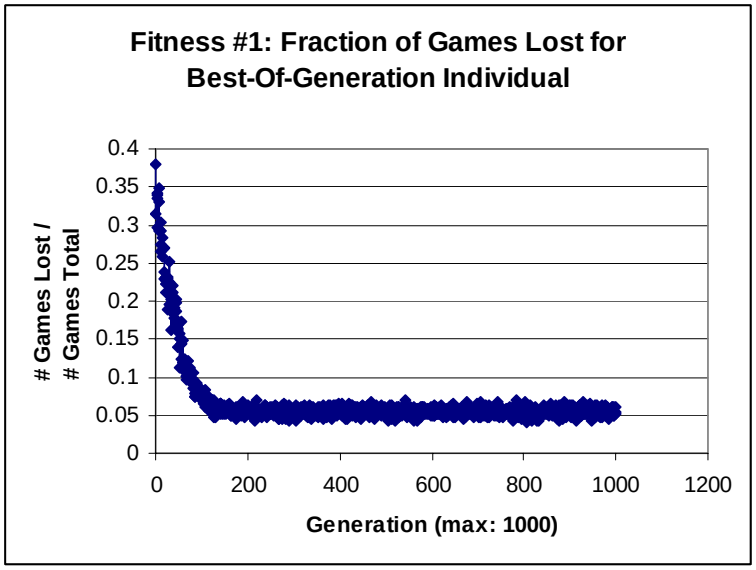


Figure 6. Best-of-Generation Performance using inadequate fitness measure

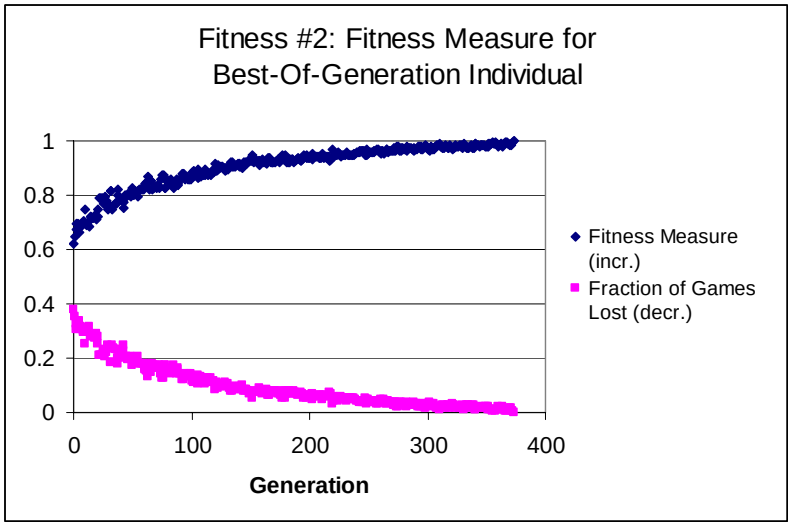
Population Size	10,000
Generations	1000
$p_{crossover}$	0.15
$p_{replicate}$	0.10
p_{mutate}	0.001

Fitness #2: Success

With the new evaluation method in place however, already the first run was successful. A population size of 500 individuals was sufficient given a maximum running time of at most 500 generations; an optimal individual was found after 1688 seconds (28 minutes), in generation 373, illustrated in Figure 7.

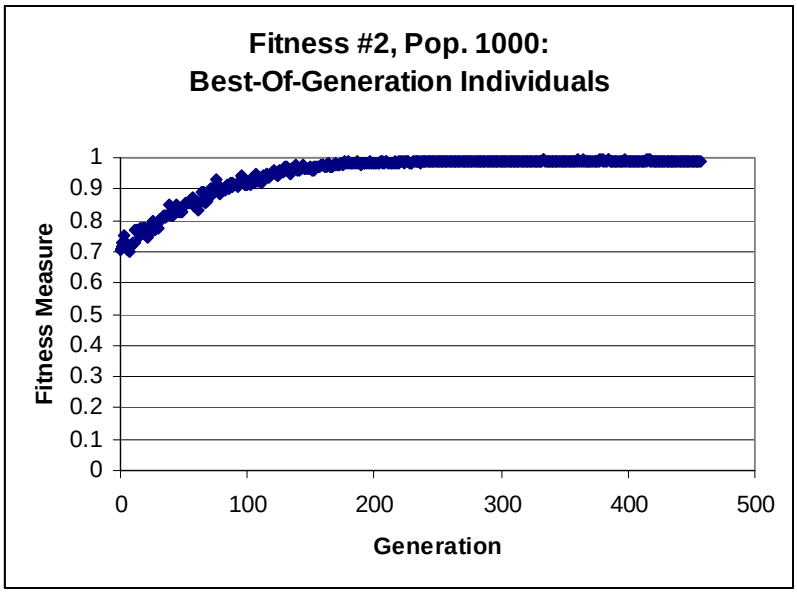
Another comparison between the two fitness measures revealed that the fittest individuals using the first method played many more matches than their counterparts with method two. The first truly maximized the number of games won and tied and the average number of games per exhaustive set was around 480 (see *Strategy Fitness Evaluation* for recalling the definition of exhaustive set). With the second measure however, the best individuals only played 320 exhaustive games on average. They tried to minimize the fraction of games lost, and by playing fewer games overall, fewer of them had a chance to be lost.

Using the new fitness measure, additional experimentation was done to determine if different population sizes and probabilities would notably affect the performance of the optimal individuals found (more games, more games tied, etc.). As it turned out, a second run with 1000 individuals did not perform better or faster; in fact its optimal individual only appeared in generation 457 (65 minutes), seen in Figure 8.



Population Size	500
Generations	termination at 373 (max: 500)
$p_{crossover}$	0.15
$p_{replicate}$	0.10
p_{mutate}	0.001

Figure 7. Successful Run using improved fitness measure



Population Size	1000
Generations	termination at 457 (max: 500)
$p_{crossover}$	0.15
$p_{replicate}$	0.10
p_{mutate}	0.001

Figure 8. Additional Run using second fitness measure with larger population size.

One more experiment varied the crossover probability to 0.3 with different population sizes. Only one out of several runs produced a perfect strategy, and only at a very late stage: using 1000 individuals, an optimum appeared in generation 692 after an extended period (96 generations) at a local maximum. All other runs however showed no optimal individual after even 1000 generations, although their best individuals were close to being perfect (1 or 2 games lost out of 300 on average). Nevertheless, the higher crossover probability was likely to cause excessive mutation in the offspring and may therefore have prevented the populations from reaching optimality in the end.

Conclusion

This paper has demonstrated how a genetic algorithm could be applied successfully to evolve a perfect Tic-Tac-Toe strategy. By taking into account board symmetry, the number of distinct game situations could be drastically reduced to 827, for which GA individuals could be optimized to adapt perfect game-playing strategies.

It was interesting to observe the adverse effects of an ineffective fitness measure in evolving the population towards a desired optimum. In the course of the experiment, the original fitness measure had to be replaced because of repeated failures, namely local maxima.

As a result, immediately the first run of the updated algorithm produced an optimum strategy after 373 generations. Despite the large genome of length 827, only 500 individuals were required. A series of additional runs confirmed this finding, and the individuals they produced did not lose a single game out of all possible matches they could ever play.

Variations in population size did not show any improvement in the fitness of the final optimal individual; and experimentations with different parameters, such as a higher cross-over rate, only resulted in longer runs or no optimum at all.

Future Work

Having successfully applied genetic algorithms to the problem of strategy evolution, more work can certainly be done to investigate the use of genetic programming in this area. While the approach taken by this project was based on a static situation-to-move mapping, future ones should consider the evolution of a Tic-Tac-Toe heuristic.

Furthermore, evolutionary programming should be applied to games of far greater complexity given its convincing performance for these lesser cases. Investigating for example the evolution of intelligent chess players by means of these concepts is certainly a motivating task to consider.

Acknowledgements

I would dearly like to thank John Koza for his teachings, suggestions, and encouragement.

References

Grefenstette, John J. September 1994. *GENESIS* (Implementation of Genetic Algorithms Framework for the C Programming Language). Available from http://www.aic.nrl.navy.mil/galist/src/

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

Schaefer, Steve. 2002. *How many games of Tic-Tac-Toe are there?* Available from http://www.mathrec.org/old/2002jan/solutions.html.