University of California Irvine
CompSci 271: Introduction to Artificial Intelligence, Fall 2014
Professor: Dr. Kalev Kask
Programming Project: Tic-Tac-Toe playing strategies for AI
Julian Collado-Umana 84562322
Pedro Silva 38567016

1) Abstract:

In this report we implemented and measured the performance of several strategies for the children's game Tic-Tac-Toe by making them play against each other. Two kinds of strategies were implemented: minimax strategies which make a decision based on all possible future moves and rule of thumb strategies which have a human-like behavior and choose based only on the closest levels of the tree.

2) Introduction:

The game of Tic-Tac-Toe is a two-player adversarial game. It consists on a 3X3 grid in which players "X" or "O" alternate marking entries on the grid. The player who places three markings in a horizontal, vertical or diagonal line in the grid wins the game.

The game has been solved, which means there is a finite number of moves and there exists a perfect strategy for the game in which the only possible outcome is a tie or a win. Because of this, games between experienced players usually end up in a tie since they reach a good enough strategy to defend in most cases. Because of this it is predominantly played among children who are still developing their strategy and it has been used to study strategy learning in human beings.[1]

In this report we explore different strategies for playing this game by simulating games using every strategy against every other one and analyzing the results. On one side human-like strategies which are basically rules of thumb given by experience. One of this will be the perfect player strategy. On the other side we use a classic search based method to decide the next move. In our case it will be the minimax algorithm with variations in case of a tie to define different strategies.

3) Strategies:

3.1) Minimax strategies

The minimax algorithm finds the optimal strategy by brute force. It first generates the whole game tree. Then it applies a utility function to the leaves. In this case the utility function will assign one point for a win, zero for a tie, or minus one for a loss. Then back-up the values to the root alternating between a max node, which chooses the max among it's children, and a min node, which chooses the min among it's children, until we reach the root which must have a max node assigned. Then search the game tree in depth first search to find the value of the root. The good things about this algorithm is that it is complete since our tree is finite and optimal. It also has time complexity O(b^m) with m=9 for our case

[1] Crowley, K., Siegler, R. *"Flexible strategy use in young children's Tic-Tac-Toe"*, Cognitive Science 17, 531-561 (1993)

and a branching factor b=9,8,7, ..., 1 as we go down the tree. Also the DFS has space complexity O(bm). In our case we have to store the whole tree so the tree size is 9! in space. This is a big value but a modern computer can handle it without problems.

The idea of minimax is that when it is my turn I choose the move that maximizes my result among all my choices and when it is my opponent's turn it will choose the move that damages me the most. Because of this the minimax algorithm plays the optimal strategy. We made slight modifications to the algorithm In case of ties between paths. Then the move is chosen based on a defined strategy. The following modifications choices are implemented as different strategies:

a) Use most wins in the tree below as tiebreaker when choosing between two equal paths.

b) Use closest wins in the tree below as tiebreaker when choosing between two equal paths.

c) Choose the path with the most wins in your next two turns in the tree below as tiebreaker when choosing between equal paths. This heuristic should give priority to moves that create forks. A fork is defined as a move that creates two or more possible winning positions for your next move. It is a common move that marks the difference between a new and an experienced player.

D.1) This strategy uses minimax only. In case of a tie it chooses the first node that was created when building the tree.

D.2) This strategy is D.1 but we also add modification "a" as tiebreaker.

The idea here is that if a path has a more possible wins in its tree it is probably a better path than one that has fewer wins.

D.3) This strategy is D.1 but we also add modification "b" as tiebreaker.

Here we choose the closest possible win in ties, this is to simulate the idea of a player that is more impulsive.

D.4) This strategy is D.1 but we also add modification "c" as tiebreaker. In the case where the game is so advanced that this search is not possible, in other words, there are less than 4 openings on the board, it D.4 automatically uses D.3 as to decide its movement.

3.2) Rule of thumb strategies:

In these strategies we select our next move based on a priority sequence defined by the strategy. The order and moves in this sequence is what defines the strategy. We first try to execute movement number 1. If it is not possible we try number 2 and so on. Notice that it makes its decision without the need to explore the whole tree. In most steps a decision can be made only checking the current board. This is it has time complexity O(b) and space complexity O(b). This makes these strategies incredibly efficient but it may be hard to come up with a good heuristic for the strategy.

S.1) Kid[2]

1) Win by placing three in a row
2) Block an opponent's win move
3) Random mark

This simulates what a new kid playing the game would do. The first strategy it would use is “Win the game” if there are already two in a row, “Block the opponent's next turn win”, and “Play randomly:” otherwise.

S.2) Smarter Kid

1) Win by placing three in a row
2) Block an opponent's win move
3) Mark center
4) Mark empty corner
5) Mark empty side

This simulates a smarter kid who has realized not all moves are equal and he should give priority to the center, then corner, then side.

S.3) Older Kid

1) Win by placing three in a row
2) Block an opponent's win move
3) Create a fork
4) Mark center
5) Mark empty corner
6) Mark empty side

This is a kid with more experience. This one has realized the importance of the fork tactic and tries to implement it whenever it is possible.

S.4) Perfect Player Strategy[3]:

1) Win by placing three in a row
2) Block an opponent's win move
3) Create a fork
4) Block fork possibility from an opponent
5) Mark center
6) Mark opposite corner from an opponent's mark
7) Mark empty corner
8) Mark empty side

[2] Based on a heuristic from: Crowley, K., Siegler, R. *"Flexible strategy use in young children's Tic-Tac-Toe"*, Cognitive Science 17, 531-561 (1993)
[3] Based on a heuristic from: Crowley, K., Siegler, R. *"Flexible strategy use in young children's Tic-Tac-Toe"*, Cognitive Science 17, 531-561 (1993)

This is the perfect player strategy. Notice how it not only implements fork, but also searches and blocks an opponent's chance to perform one.

4) Implementation:

The project was implemented in Python 3.4.1 and all the tests were performed in a MacBook Pro, OS X version 10.10, processor Intel Core i7 2.5 Ghz, and 16 GB of ram memory. With this amount of memory we expected to eliminate any possible space issues, since we have to create a tree that can have over 300.000 nodes.

A “Node” on the tree is a structure that holds several other structures: depth: indicates in what depth the node is, board: a list with nine positions representing the board, turn: indicates if the current node is in a maximizing or minimizing turn, parent: reference to the parent node, state = strategy value associated with the node, action: a tuple of two values representing the action performed from the last node to the current one, children: a list of children nodes, value: value used to analyze a node for the maxmin algorithm, and game_result: indicates if the current nodes represents a win, a loss, a draw, or none of them.

Although most of the games had reasonable time executions, the games played by the any of the “D” strategies were slower if compared to the “S” ones, which was expected since a big portion of the tree has to be checked every time a decision has to be made. Additionally, a queue was used to hold the nodes to be analyzed on the “D” strategies. This is the case because these strategies use non-recursive breadth search and depth search, and the use of a queue is a common practice to make this implementation.

Among these, D.2 showed the worst performance regarding time of execution. When D.2 starts it has to use its tiebreak strategy on 8 out of 9 sub-trees. This task takes a prohibitive amount of time for a real-time game: around 10 to 15 minutes.

In the implementation submitted the “Main” class allows the user to play against one of the strategies described or to create a game between two strategies. To create the table 1 we created another main function, depicted on the “Main2” class, which runs all the strategies against each other.

5) Results

We simulated several games between the agents following the different strategies and we obtained the results seen in table 1.

Simulated games results									
		N W / L / D	N W / L / D	N W / L / D	N W / L / D	N W / L / D	N W / L / D	N W / L / D	N W / L / D
		D.1	D.2	D.3	D.4	S.1	S.2	S.3	S.4
D.1	F	3 0 / 0 / 3	4 0 / 0 / 4	4 0 / 0 / 4	5 0 / 0 / 5	50 46 / 0 / 4	2 0 / 0 / 2	6 0 / 0 / 6	5 0 / 0 / 5
	S	7 0 / 0 / 7	6 0 / 0 / 6	6 0 / 0 / 6	5 0 / 0 / 5	50 7 / 0 / 43	8 0 / 0 / 8	4 0 / 0 / 4	5 0 / 0 / 5
D.2	F		5 0 / 0 / 5	6 6 / 0 / 0	3 0 / 0 / 3	3 2 / 0 / 1	6 6 / 0 / 0	5 0 / 0 / 5	5 0 / 0 / 5
	S		5 0 / 0 / 5	4 0 / 0 / 4	7 0 / 0 / 7	7 0 / 0 / 7	4 0 / 0 / 4	5 0 / 0 / 5	5 0 / 0 / 5
D.3	F			7 0 / 0 / 7	6 0 / 0 / 6	47 39 / 0 / 8	4 0 / 0 / 4	4 0 / 0 / 4	6 0 / 0 / 6
	S			3 0 / 0 / 3	4 0 / 0 / 4	53 10 / 0 / 43	6 0 / 0 / 6	6 0 / 0 / 6	4 0 / 0 / 4
D.4	F				7 0 / 0 / 7	45 39 / 0 / 6	3 3 / 0 / 0	6 6 / 0 / 0	4 4 / 0 / 0
	S				3 0 / 0 / 3	55 12 / 0 / 43	7 0 / 0 / 7	4 0 / 0 / 4	6 0 / 0 / 6
S.1	F					46 18 / 10 / 18	54 5 / 4 / 45	58 8 / 8 / 42	46 4 / 3 / 39
	S					54 9 / 14 / 31	46 0 / 19 / 27	42 0 / 24 / 18	54 0 / 33 / 21
S.2	F						53 0 / 0 / 53	54 0 / 0 / 54	49 0 / 0 / 49
	S						47 0 / 0 / 47	46 0 / 0 / 46	51 0 / 0 / 51
S.3	F							43 0 / 0 / 43	42 0 / 0 / 42
	S							57 0 / 0 / 57	58 0 / 0 / 58
S.4	F								40 0 / 0 / 40
	S								60 0 / 0 / 60

Table 1. Simulated game results for different strategies. H is the heuristic number. F means it was a game in which it made the first move, S stands for second move, N is the number of games simulated, W is the number of wins, L the number of losses, and D the number of draws.

6) Discussion

First we have to notice some of the strategies are deterministic. This is they will always make the same moves if the opponent follows a non-stochastic strategy. This is the case of the minimax when playing one against the other. The decision is based on the result of the leaves of the whole tree, since the tree is always built in the same order it will always choose the same path if nothing else changes. In contrast we can see the first three rule of thumb strategies. This performs a random choice in some cases and as such they can give different results each time we run them. This is the main reason for the asymmetry in the number of simulated games. The other reason is D.2 takes a very long time to make a decision because it is common that there exist ties in the minimax, and then it has to explore the whole tree

again every time it has to make a move to count the number of wins in each subtree. When D.2 starts a game for example, 8 out of the 9 possible movements have to be further explored in order to come up with a choice.

We can see that whenever the minimax strategies play against themselves the result is a draw. This is the expected result since the minimax algorithm will give the optimal strategy and a draw is the result of playing an optimal strategy in both sides.

We see more interesting results when comparing the minimax strategies with the rule of thumb strategies. For example when playing against S.1, the starting kid strategy, the usual result is a loss for S.1 when it plays second and a draw when it plays first. However, when S.1 starts we see that in a about 20% of the cases the match ends in a draw. Since S.1 is almost playing randomly this leads us to think achieving a draw in this game isn't really that hard. We also see the results shift when S.1 does not start. Now 20% of the games end in loss for S.1 and the rest in a draw. This means starting the game has an incredibly important role in the result of the match, making it in this case much easier to reach a draw.

Remember the minimax strategies play a perfect game, which is supposed to end in a draw or win, but they are waiting for their opponent to make a mistake and then win. The hard thing to do in this game then is to reach a result that is not a draw.

We see another interesting result when looking at the outcomes S.2's games. In general again we reach a draw in all the matches where minimax was the second player. However in the matches where D.2 and D.4 started, S.2 loses consistently instead of reaching a draw like it did against D.1 and D.3. Since all D strategies are just waiting for their opponent to make a mistake, this means D.2 and D.4 are better at creating a game in which S.2 will make a mistake than the other D strategies. This result is very useful. It tells us D.2 and D.4 would probably perform better against not so experienced human players than the other minimax strategies. In other words, their tie breaking mechanism is an improvement in the algorithm for this game when playing against humans.

When comparing the results for the D strategies against S.3 we see a similar result as with S.2. The difference now is that in this case D.4 is the only one that achieves a win, and does so consistently. This again suggests D.4 will give a better performance when playing against expert humans.

Then we analyze the results against the "perfect" strategy and we find a surprise. It doesn't reach a draw in every game as we expected. It actually loses against D.4 in every case when D.4 starts and it loses sometimes against S.1, the kid! This means our implementation of this strategy is not really perfect and D.4 is very good at creating the trap for almost perfect human-like players to fall in. We will have a further discussion on this.

For S.1 against itself we see again starting gives a big advantage and it leads to more wins while starting second leads to more draws and the number of loses is pretty much the same. This result is expected since most of the moves would be random and a win could be achieved by generating a random fork, otherwise the predominant result should be draws because of the Block movement.

When S.1 plays against S.2 we can see S.2 is a lot better as we expected by giving priority to better moves. This results in S.1 rarely winning in case it begins and no wins when it is the second player with

the predominant result being a draw. This indicates S.2 is a better strategy than S.1 but still it could improve.

For S.1 against S.3 we can see predictable results. S.3 loses very few times and wins or ties in most of the games. S.3 also performs slightly better than S.2 as we would expect by the introduction of the Fork move which should result in more wins since it lays a trap. We see our results are consistent with this assumption, thus indicating Fork is a good tactic to include in our strategies. We can see here these strategies prefer certain types of movements over others (center>corner>side) but it would be interesting to test other combinations to see how they perform.

However the improvement from S.2 to S.3 is very small and we see when they play against each other or against themselves they end up in draws every time. A close analysis of the final boards reveal the Fork move is never executed in these games. This means the opportunity of creating a fork is never available in the combination of moves made by choosing center>corner>side. This is in contrast to the game with S.1 where the Fork move is used on purpose by S.3 and by randomness generated by S.2.

When comparing any strategy against S.4, one of the perfect strategies, we see it leads to a draw every time as expected except against S.1 in some rare cases. Here S.4 effectively blocks the Fork movement for the opponent, and since this is the only way of winning because of the Block movement, then every match ends up in a draw or a win for S.4. What happens is basically that S.4 waits until the opponent makes a mistake, and then wins. We were, again very surprised to see S.1 beats S.4 sometimes. This happens because of the high randomness in S.2 strategy as we discovered.

We traced the games to find how S.1 was winning against the "perfect" strategy. We found this game when S.1 starts playing with O:

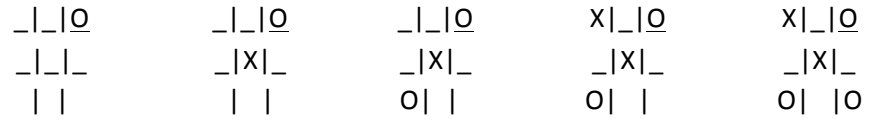

Figure 1. Sample of game between S.1 and S.4 in which S.1 unexpectedly wins.

What we see is S.1 (O) is creating a double fork opportunity for the next turn because of symmetry. Even if the fourth move (X) is in a corner to block the fork, the other corner is still available to create another fork. Our Block fork strategy is able to detect the fork possibility and block it, but it is not able to detect multi-fork moves. Since this strategy was adapted from a reference it means their block fork move takes into account more cases than ours. We defined block fork as a search for a possible fork opportunity from the opponent in their next turn, and if found prevent them from marking that position by marking it ourselves, as seen in figure 1. However to effectively block this double fork, instead of playing in the corner in move number four, it should play a side in order to force the O player to Block instead of creating a fork and the game ends in a draw. This is the same strategy used by D.4 when playing. The interesting part is that D.4 is better at creating this trap for the other player to fall into than the other minimax strategies.

This surprising result also suggests that a corner is in these cases a better starting move than the center as we initially though and most human players think as well. In this case randomness was able to beat

the "perfect" strategy in a Monte Carlo like debugging game. We wonder for future work if it would be computationally feasible to reach the perfect strategy using Monte Carlo and if it would be different from the one we have, this is if there is more than one perfect solution.

Summarizing, the minimax strategies are in general perfect, but some achieve better results than others when playing against human-like players. This is because they are waiting until the other player makes a mistake and some of them seem to be better at setting traps in which humans tend to fall into. We achieved these improvements in minimax by changing the way decisions are made in case of a tie in the values of the children nodes. We also discovered our "perfect" strategy (S.3) was actually missing one case and this was discovered as a consequence of randomness first and then when it was playing against our best minimax strategy (D.4). Also, implementing a few rules in our strategy like prioritizing some movements gives us a very good defensive improvement as seen when comparing S.1 with S.2, and S.3. However these would still fail when playing against a clever human being if they start second as seen in figure 1. We expect an improved version of the "perfect" strategy will be able to defend against this move and indeed be perfect like D.4 but without the need of creating the whole tree.

7) Conclusions

Perfect strategies like the ones followed by minimax will be able to achieve in worst case scenario, a draw. However the real improvement when comparing perfect strategies lies in being able to trick the other player into making a mistake and then winning. In this sense using strategy "c" as tiebreaker in minimax gives the best results and the runner up is strategy "a" while "b" didn't show an improvement.

Starting the game gives a big advantage no matter what strategy we use.

There seem to exist preferential positions: Center seems to be better than corner which seems to be better than side in most cases. One exception is in the first move of the game as seen in figure 1.

Creating a Fork is a great advantage and it is what leads to winning when playing against more advanced players. Creating a double fork is the only strategy to beat expert and almost perfect players.

Randomness is a great mechanism to find weak points in strategies as seen in the results of S.1 against S.4.

Achieving a draw in Tic-Tac-Toe doesn't seem to be too hard, especially when playing first.

8) Future work

Try other combinations of preferential movements to see if they lead to an improvement, for example Corner>center>side.

Measure the performance of the a new corrected "perfect" strategy which starts playing a corner instead of center.

Use Monte Carlo to reach a perfect strategy.

Different implementations of the BFS and DFS used on the “D” strategies can be tested to see if makes the execution of those faster.

9) References

-Crowley, K., Siegler, R. *"Flexible strategy use in young children's Tic-Tac-Toe"*, Cognitive Science 17, 531-561 (1993).
-Russel, S., Norvig, P. "Artificial Intelligence: A Modern Approach", 3rd ed., Prentice Hall, 2009.