Decentralized Decision Making in the Game of Tic-tac-toe

Edwin Soedarmadji, *Student Member, IEEE*

Abstract — **Traditionally, the game of *Tic-tac-toe* is a pencil and paper game played by two people who take turn to place their pieces on a 3×3 grid with the objective of being the first player to fill a horizontal, vertical, or diagonal row with their pieces. What if instead of having one person playing against another, one person plays against a team of nine players, each of whom is responsible for one cell in the 3×3 grid? In this new way of playing the game, the team has to coordinate its players, who are acting independently based on their limited information. In this paper, we present a solution that can be extended to the case where two such teams play against each other, and also to other board games. Essentially, the solution uses a decentralized decision making, which at first seems to complicate the solution. However, surprisingly, we show that in this mode, an equivalent level of decision making ability comes from simple components that reduce system complexity.**

I. INTRODUCTION

PERHAPS it is not an exaggeration to claim that the game of Tic-tac-toe is among the most popular childhood games in the world. The game is played by two players who place their different colored or shaped game pieces on a 3×3 grid. Unlike checker, chess, weiqi (go), and many other board games, the relatively simple grid enables people since antiquity to play Tic-tac-toe on beach sands, napkins, dusty windshields, or wherever the grid can be drawn. The rule is very simple: players take turn, each time placing one of their pieces in an unoccupied position on the grid until the grid is filled, or until someone wins. The objective is also easy to understand: the first player to fill a horizontal, vertical, or diagonal row with his / her pieces wins the game.

The fact that Tic-tac-toe is so simple and widely known makes it the ideal game of choice for classroom introduction to programming, game theory, data structure, and combinatorial enumeration of all possible game outcomes. By one account, there are 765 essentially different configurations of the game pieces, which translate into 26,830 possible games, taking into account different symmetries. If symmetry is not considered, in total, there are 255,168 possible games.

While 255,168 sounds like a large number for a human player to memorize, it is certainly not a large number for most modern computers. One can imagine "training" a computer to be a competent player in Tic-tac-toe by memorizing all 255,168 games and use this knowledge to calculate the best move based on the existing board configuration.

This approach is of course a far cry from how people play. No one memorizes all 255,168 games to calculate the best move. Instead, we humans use simple rules that provide several possible moves, from which the best move was chosen. Sooner or later, most human players discover that the game of Tic-tac-toe always ends in a draw when both players know and use the optimal rules of the game.

Just as we humans develop our decision making ability through playing games — starting from the simple and concrete games to the more complex and abstract games — in its evolution, computers spent some of their "childhood" years playing Tic-tac-toe (ironically, after being forced into the horror of calculating ballistic trajectories, code breaking, and simulating atomic explosions in its "infanthood" years!)

One of the first computers in the 1950's to play Tic-Tac-Toe, EDSAC1 was capable of playing a perfect game with a program less than 4,000 bytes long. A human played against a single player, the machine. This tradition continued well into the modern era of Internet. A casual search on the Internet would return hundreds, if not thousands, of interactive web pages capable of playing perfect games against human players. Taking a peek into the source codes behind these games and stripping away the user interface codes — leaving behind only the equations, logic, and database used by the programs — one cannot avoid the impression of relative complexity for such a simple game. Can we do better?

This paper addresses the interesting question: is there a simpler way to program a competent Tic-tac-toe player? Can complexity be further reduced by a different, i.e., decentralized mode of decision making? Later, we elaborate the definition of a competent player. For now, we simply mean a player who makes no mistake and can consistently force a draw, regardless of which player starts the game.

What if instead of trying to create one monolithic competent player, we create nine agents and one manager, together acting as a single competent player? At first, it sounds like we have increased the complexity of our solution. After all, the team still has to respect the original rules of the game, meet the same priorities, and on top of that, coordinate its action. However, in this paper we show that surprisingly, the end result is a simpler set of rules for each agent. Consequently, the team has a much lower complexity compared to an equivalent centralized implementation, as evidenced by the types and numbers of instructions used by the team.

Further, it is not hard to imagine that in certain computing platforms, decentralized decision making is the only possible avenue of computation. In the next section, we begin the presentation by analyzing a competent Tic-tac-toe player.

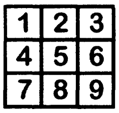
Manuscript received December 18, 2005.

E. Soedarmadji is with the California Institute of Technology,
Pasadena, CA 91125 USA. edwins@systems.caltech.edu.

This work was sponsored by the Lee Center for Advanced Networking.

II. COMPETENT PLAYER

A competent player is defined as a player who has in its arsenal a complete collection of strategies that are necessary to consistently force a draw when faced with another competent player or win when the other player makes a mistake. In contrast, a less-than-competent player only has a subset of these strategies. For convention, the grid is numbered as in Figure 1 below. In this paper, we assume that the opponent plays the O (for "opponent") while the team plays the X.



1	2	3
4	5	6
7	8	9

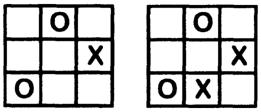
Figure 1 Grid cell numbering convention

Based on which defensive strategies a player has, we can categorize the players into the following player categories:

A. *Defensive Strategic Categories*

A *novice defensive player* can block an immediate threat, i.e., it can prevent an opponent from completing a row. Such a player detects the presence of two opponent pieces on a row and reacts by placing its own piece in the remaining space on the targeted row.

An *intermediate defensive player* can detect and preempt any attempt by the opponent to introduce two possible completions. For example, in Figure 2, suppose the opponent, player O, just moved. An intermediate defensive player would know that the right move is to prevent the opponent from occupying cell 8, thus preventing a two-completion attack on cell 5 and 9 in the next move.



	O	
		X
O		

	O	
		X
O	X	

Figure 2 Intermediate defensive player

However, note that the opponent can also launch a two-completion attack on cell 3 and 4 by occupying cell 1. To force a draw, the player needs to utilize more than defensive moves. As the famous dictum says, "The best defense is offense." If we occupy cell 5, the opponent is forced to follow a series of defensive moves that lead to a draw.

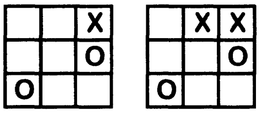
An *advanced defensive player* can react appropriately to an opening move. In Tic-tac-toe, this translates to placing a piece in the center cell if the opponent starts anywhere but the center. If the opponent starts from the center cell, such a player reacts by occupying one of the corner cells.

Similarly, we can categorize players based on their offensive capabilities, which are listed below.

B. *Offensive Strategic Categories*

A *novice offensive player* can complete a row when two of its own pieces are already placed in a row with one remaining empty position. This is, as the name suggests, the most basic offensive capability a player must have to have a chance of effectively winning against another player.

An *intermediate offensive player* can threaten the opponent with a one-completion attack, thus forcing the opponent to react and defend the position, possibly disrupting any planned move. For example, in Figure 3, suppose the opponent just placed an O in cell 6. An intermediate offensive player would threaten the opponent by placing his piece in cell 1 (or 2), forcing the opponent to occupy cell 2 (or 1).

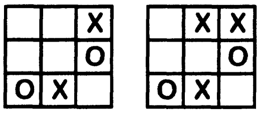


		X
		O
O		

	X	X
		O
O		

Figure 3 Intermediate offensive player

An *experienced offensive player* can threaten the opponent with two possible completions on two rows, thus guaranteeing a win. For example, in Figure 4, suppose the opponent just placed an O in cell 6. An experienced offensive player would force a win by placing an X in cell 2, threatening a two-completion attack in cell 1 and 5.



		X
		O
O	X	

	X	X
		O
O	X	

Figure 4 Experienced offensive player

Finally, an *advanced offensive player* can make the most aggressive opening move. Playing against an opponent executing random moves, this means occupying any one of the corner cells, giving a 7 out of 8 chance of winning. Playing against a competent opponent, the most strategic move is to occupy the center cell, denying four possible completions.

III. TEAM INFRASTRUCTURE

Having described the different player categories, of course we eventually want to create a team of 9 agents (plus one manager) that can emulate, through their independent actions, the level of competence shown by an advanced defensive and offensive player. However, first let us describe the infrastructure available to the team members.

First, let us describe the role of the manager as shown in the algorithm below. It performs a primitive coordinating role for the agents and nothing more. In fact, the same manager can be used for Tic-tac-toe or other turn-based (could be multi-player) board games because the manager knows almost nothing about the game its agents are playing.

```
MANAGER
1: Wait until a new opponent piece is placed.
2: Once placed, ask all active agents to start calculation.
3: Wait for the agents to submit their responses.
4: Choose one of the responses with the highest priority.
5: Notify all agents which agent is selected.
6: If all cells are filled, then end. Otherwise, go back to 1.
```

Each agent responds back with a priority number, essentially saying "let me handle this." In step 4, the manager selects the agent with the highest priority. If there is a tie, it selects one response (either at random or first arrival, etc.)

These priority numbers convey the subjective and private belief of each agent of how important it thinks the information it has, and the reaction it plans to do, to the success of the team (in this case, in playing the Tic-tac-toe game, although this could be easily extended to other applications!) The manager thus resolves any possibly conflicting subjective views by impartially (or partially, in a consistent way) selecting one of the agents. Let us now describe the role of the agents, as shown in the algorithm below:

```
AGENT
1: Wait for the instruction to start from the manager.
2: If the opponent already landed in this cell, or reaction
   is already made, then end. Otherwise, move to 3.
3: Obtain all accessible information about the board.
4: Consult the function T for a priority number.
5: Once found, submit the priority number as a response.
6: Wait for the selection notification from the manager.
7: If selected, then react. Otherwise, go to 1.
```

Before explaining the algorithm, let us distinguish the word "response" and "reaction". An agent *responds* to the manager by providing a priority number. In contrast, an agent *reacts* to the opponent by placing a friendly piece where the agent is assigned to operate.

Step 1 is trivial. It simply asks the agent to wait for the instruction from the manager before it begins calculating. This is important because in order for a calculation to be reliable, it has to be done based on the most current and relevant state of information available. The manager has a global knowledge of when the opponent introduces a new piece into the board. Therefore, step 1 provides the synchronizing signal for information processing that precedes decision making.

Step 2 is also easy to understand. An agent no longer has to compute a reaction if it has already made one, or if the opponent has eliminated any reason for the agent to make one (by placing its piece where the agent is located.)

Step 3 is very crucial to an agent's operation. In one extreme case, an agent might calculate a response in absence of any factual information of the board configuration, i.e., it is simply a "fortune-teller" – providing suggestion to the manager based on its internal and unsubstantiated private beliefs. In another extreme, an agent has complete information on the board configuration. It is easy to say that both modes of information access are not desirable (or practical).

What makes our model interesting is the case where the agents have incomplete information (by design) about the board, from which they infer the best move in their own areas of responsibility. The agents then convey this inference to a manager, who then arbitrates conflicting priorities.

In what will become clear in the imminent examples in this paper, the agents do not have to access the same rule, level, scope, and type of information. In many simple board games, the agents can be identically programmed. However, in more complex games, the agents can easily operate within an informational and operational (rule) hierarchy.

Step 4 essentially declares the existence of a "rule book" for each agent. Of course, in some games, the agents can also be permitted to "improvise", i.e., suggesting certain actions and priorities based on their own internal probabilistic process unknown to the manager. Confronted with a board scenario (which is nothing more than the information about the board available to the agent), the agent attempts to judge, "how important is my reaction going to be compared to those of other agents?" The answer is scored by its priority number and then submitted to the manager in step 5.

Finally, in steps 6 and 7, the agent simply waits for the manager's response. If the manager decides to activate the agent, then the agent reacts and fulfills its mission. If not, it waits for another round of decision making.

IV. TIC-TAC-TOE TEAM

The team infrastructure described in the previous section allows us to start our construction of a competent Tic-tac-toe player by first building a novice defensive team that perfectly emulates a novice defensive player.

Let us assume that there are three types of agents, each with their own programs and level of information access. Therefore, there are three types of functions *T* used in step 4.

In case of Tic-tac-toe, the information access rule is such that "an agent has perfect information on the states of all cells located in the same horizontal and vertical (and whenever appropriate, diagonal) row as the cell it is in". The state could be empty, occupied by an opponent piece, or by a friendly piece. Each agent knows the cell it occupies. In Figure 5, the agent is marked by an X, and the cells to which it has perfect information is marked by the bullet symbols.

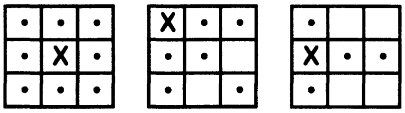


Figure 5 Information access rule

The agent located in the center cell of the grid must have what amounts to a perfect information on all the cells (see the left box). The agents in the corner cells must have access to the horizontal, vertical, and diagonal rows (6 cells – see the middle box), and the agents along the edges only have to know the horizontal and vertical rows (4 cells).

```
if n(oo) ≥ 1 then return 1
return 2
```

Figure 6 Novice defensive strategy

We claim that for a novice defensive team, the function *T* can be as simple as the one shown in Figure 6. The notation `n(oo)` means the number of horizontal, vertical, and diagonal rows containing two opponent pieces. In this notation, a friendly piece is denoted by an `x`, and a blank cell is by a `b`. For example, `n(bx)` means the number of rows containing a blank and a friendly piece. Evaluated at the center, corner, and edge cells, the function `n(·)` can return up to four, three, and two rows, respectively.

Suppose the board configuration is shown in Figure 7a. In this game, the opponent pieces are marked by the O's. Figure 7b shows the priority numbers calculated by all the nine agents as they are submitted to the manager. The team will correctly block the attack by occupying cell 1.

	O	O
		X
X		

1	O	O
2	2	X
X	2	2

Figure 7 Novice defensive play

Of course, the opponent O could be smarter and present a two-completion attack as shown in Figure 8 below. In this scenario, all agents in a novice defensive team report a non-severe priority number of 2. The manager thus randomly (or systematically) selects one of the 5 possible agents, and only with a probability of 1/5, the manager selects the agent in cell 3, which directly neutralizes the two-completion attack.

O		
X	X	O

O	2	2
X	X	O
2	2	2

Figure 8 Failure of the novice defensive strategy

How do we prevent this uncertainty? One obvious solution is to put more smarts into the team – and since the manager is dumb, this means making the agents smarter. Instead of the *T* shown earlier, the agents can use a more robust *T*:

```
if n(oo) ≥ 1 then return 1
if n(ob) ≥ 2 then return 2
return 3
```

Figure 9 Intermediate defensive strategy

The function gains another line, but the agents can now collectively defend against two-completion attacks from the opponent! Now, if the agents are confronted with a scenario shown in Figure 10 below, they independently evaluate priority numbers shown in the right subfigure, and the manager correctly chooses the best agent to fend off the attack.

O		
X	X	O

O	3	2
X	X	O
3	3	3

Figure 10 Intermediate defensive play

In Figure 8, the agents in cells 2 and 8 can also thwart the two-completion attack by launching a high-priority attack on the opponent, rather than neutralizing the opponent's lower-priority two-completion attack. But this requires more than defensive strategies, but also some offensive capabilities. By process of induction one infers that *T* needs to be expanded further and this is indeed correct. To emulate a novice offensive player, the agent now also has to have access to information on friendly pieces on the board.

However, now the question is which one should have a higher priority: the novice offensive strategy (which basically ends the game with a win) or the novice defensive strategy (which averts a loss by another move)?

Obviously, anything that ends the game with a win takes higher priority. This principle is reflected in the version of *T* implementing offensive capability shown in Figure 11.

```
if n(xx) ≥ 1 then return 1
if n(oo) ≥ 1 then return 2
if n(ob) ≥ 2 then return 3
return 4
```

Figure 11 Novice offensive strategy

If an agent is still making this calculation, then the agent itself has not made a reaction and is located on a blank cell. Armed with an offensive strategy, the agent can therefore win the game for the team by making a reaction. For example, suppose the X team is confronted with the board configuration shown in Figure 12 below. Using the above function *T*, the agent in cell 9 reacts and secures the win.

	O	O
	O	
X	X	

2	O	O
4	O	3
X	X	1

Figure 12 Novice offensive play

We can extend the function *T* further to implement the intermediate and experienced offensive strategies as shown in Figure 13 below. The symbol `ee` represents both `bb` and `ox`.

```
if n(xx) ≥ 1 then return 1
if n(oo) ≥ 1 then return 2
if n(bx) = 2 then return 3
if n(bx) ≥ 1 then return 4
if n(bo) = 2 then return 5
if n(ee) = 2 then return 7
if n(ee) ≥ 1 then return 6
```

Figure 13 Intermediate and experienced offensive strategies

An example of the board configuration that showcases the application of these new strategies is shown in Figure 14. In this configuration, the team is confronted with the choice of blocking a two-completion attack by occupying cell 3, or completing its own threat of two-completion attack by occupying cell 7. Using the previous function, the team correctly adopts an aggressive stance and secures the win.

	O	
X		O
	X	

4	O	5
X	7	O
3	X	4

Figure 14 Intermediate and experienced offensive play

At this point, we can claim that we almost have a team of agents (plus a manager) that emulate the ability of a competent player. The function *T* for each agent is very simple and intuitive. Although coordinated centrally, decision is made in a decentralized manner with locally available information.

We have not discussed the all-important opening move. In Tic-tac-toe, this move determines the course of the game. If the team starts first, how do we customize the function *T* such that it starts from the center? If the opponent moves first, how do we program *T* so the team responds at the center if the opponent starts from the corner and vice versa?

The answer of course lies in the function *T*. Nowhere in this paper do we require the agents to be identically programmed, i.e., they can have different function *T*! So suppose we use the following functions T_1, T_2, and T_3 for the center, corner, and edge agents, respectively:

```
T1:   return 1
T2:   if n(xx) ≥ 1 or center != x then return 1
      if n(oo) ≥ 1 then return 2
      if n(bo) = 2 and n(bx) = 1 then return 3
      if n(bx) = 2 then return 3
      if n(bx) ≥ 1 then return 4
      if n(bo) ≥ 2 then return 5
      if n(ee) ≥ 2 then return 7
      if n(ee) ≥ 1 then return 6
T3:   if n(xx) ≥ 1 then return 1
      if n(oo) ≥ 1 then return 2
      if n(bx) = 2 then return 3
      if n(bx) ≥ 1 then return 4
      if n(bo) ≥ 2 then return 5
      if n(ee) ≥ 2 then return 7
      if n(ee) ≥ 1 then return 6
```

Figure 15 Fully implemented strategy

At this point, we have reached our original objective of constructing a competent Tic-tac-toe team. In the next section, we discuss several theoretical issues and the issue of extending this framework to other types of board games.

V. DISCUSSION

Many interesting issues arise from this new framework. For example, is it even possible for the agents to initiate a two-completion attack on the opponent given that they only have access to the information from cells on the same row?

The answer is, yes, it is possible. However, it cannot be done without using indirect inference and reducing the aggressiveness of the agents. In Figure 16, we illustrate why this is so. Given the board situation shown on the left, the agents use the *T* in Figure 15, resulting in Figure 16b.

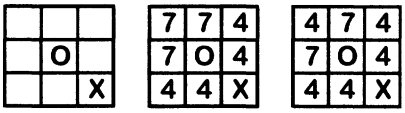


Figure 16 Initiating a two-completion attack

Using the program in Figure 15, the agents launch their attacks immediately. The corner agents can be programmed to initiate two-completion attacks if the priority number for the event `n(bb)=2` is mapped to a 4, shown in Figure 16c. Now, instead of being limited to responding with immediate attacks, the agents can launch a delayed, although more potent, coordinated attack. Thus, we can say that with this change, the overall aggressiveness of the team is reduced (although we can argue that the team's finesse is increased).

Another issue is whether the team manager can ask only a subset of the active agents that are immediately affected by the opponent's move. For example, if the opponent starts from a corner, the manager could ask the agents on the two edges and one diagonal intersecting the "invaded" corner.

Obviously, the advantage of this method is a reduced level of agent activity. Because the Tic-tac-toe grid is small, the saving is quite small. However, if we extend this method to a game with a much larger board (for example, checker), the saving can be substantial. Further, this method allows for a second level of decentralization (or a hierarchy) by introducing local managers into the game. These managers then have their own areas of responsibilities and agents.

The disadvantage does not become obvious until this decentralized team faces either a monolithic player with a superior computation capacity, or another superior team opponent not constrained by limits on agent activity and communication. Such strong opponents can devise maneuvers that provoke agents from different managerial areas of responsibility to react in separate ways that might be locally optimal with respect to the limited knowledge and coordination they have available, but nevertheless ineffective to neutralize the lethality posed by the global threat posed by the opponent.

Finally, this paper would not be complete without discussing the extension of this approach to games other than Tic-tac-toe. Games similar to Minesweeper (which has been proven to be NP-complete) can benefit from a decentralized approach — in fact, in real life minesweeping operations, this approach is the ONLY way! Reversi is another example of a game that can use this decentralized approach. In Reversi, the manager activates only those agents adjacent to the occupied cells. The information of interest to these agents is not only the number of opponent and friendly pieces along the rows, but rather the number of consecutive opponent pieces terminated by a friendly piece. Unlike in Tic-tac-toe where the center cell is the most strategic cell, in Reversi, the four corner cells can determine the outcome of the game.

VI. CONCLUSION

In this paper, we presented a decentralized method of playing Tic-tac-toe game. The method is extensible to other turn-based board games, especially those games where the pieces do not move once placed on the board. We discussed a possible extension to the game of Reversi (although a detailed implementation is beyond the scope of this paper).

There are many interesting research questions that rise from the results we presented in this paper. For example, can this method be extended to other games such as Checker, Fox and Geese, etc.? These games differ from Tic-tac-toe because their pieces move within the board. Finally, is there an automated way to create the rule table *T* for team agents, and a reliable way to verify them once created? We hope this paper stimulates the readers to answer these questions.

REFERENCES

[1] M. Gardner, "Ticktacktoe Games." *Wheels, Life, and Other Mathematical Amusements*. W. H. Freeman, pp. 94-105, 1983.

[2] Wikipedia contributors, "Tic-tac-toe", *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org [accessed 19 December 2005]

[3] M.R. Williams, "Cambridge celebrates 50 years since EDSAC." *IEEE Annals of the History of Computing*, 21(3): 72-72, 1999.