# Alloy Model of Playing Tic-Tac-Toe

I model the tic-tac-toe game board this way:

one sig gameBoard {

cells: Row -> Col -> Mark -> Time

}

Mark is either X or O:

enum Mark { X, O }

Row and Col are simply sets:

sig Row {}{ #Row = 3}

sig Col {}{ #Col = 3}

There is a winner when:

* there is a row with all X's or all O's, or
* there is a col with all X's or all O's, or
* there is a left-to-right diagonal with all X's or all O's, or
* there is a right-to-left diagonal with all X's or all O's.

I express that with the following predicate.

pred winner [t: Time] {

some m: Mark |

some r: Row | all c: Col | board[r, c, t] = m

or

some c: Col| all r: Row | board[r, c, t] = m

or

board[first, first, t] = m and

board[first.next, first.next, t] = m and

board[first.next.next, first.next.next, t] = m

or

board[last,last, t] = m and

board[last.prev, last.prev, t] = m and

board[last.prev.prev,last.prev.prev, t] = m

}

Here is my complete tic-tac-toe model:

open util/ordering[Time]

open util/ordering[Row]

open util/ordering[Col]

/\*

Structure:

1. The game is played on a 3x3 board.

2. There are two players, Player1 and Player2.

3. Players mark the game board with either X or O.

4. The game is played over a series of time steps.

\*/

// 4. The game is played over a series of time steps.

sig Time {}

// 3. Players mark the game board with either X or O.

enum Mark { X, O }

// 2. There are two players, Player1 and Player2.

enum Player { Player1, Player2 }

// 1. The game is played on a ... board.

one sig gameBoard {

cells: Row -> Col -> Mark -> Time

}

// 1. ... on a 3x3 board.

sig Row {}{ #Row = 3}

sig Col {}{ #Col = 3}

/\*

Constraints:

1. Each cell has at most one mark (X or O) at each time.

2. A win stops further marking.

3. When all cells are marked, there can be no additional marking.

4. Players alternate moves.

5. There is no interrupt in the play: If cells are empty at time t-1,

and there is no winner at time t-1, then there will be one

fewer empty cells at time t. If there is a winner at time t-1,

then there will be no change to the number of empty cells at

time t (per invariant 2).

6. Player1 marks cells O and Player2 marks cells X.

7. When there is a winner or when all cells are marked,

then the recording of "last player to move" is blank.

\*/

// 1. Each cell has at most one mark (X or O) at each time.

pred Each\_cell\_has\_at\_most\_one\_mark {

no r: Row, c: Col, t: Time, disj m, m': Mark |

((r -> c -> m) in gameBoard.cells.t) and

((r -> c -> m') in gameBoard.cells.t)

}

// 2. A win stops further marking.

pred gameBoard\_remains\_unchanged\_after\_win {

all t: Time - first |

winner[t.prev] => gameBoard.cells.t = gameBoard.cells.(t.prev)

}

// 3. When all cells are marked, there can be no additional marking.

pred gameBoard\_remains\_unchanged\_after\_every\_cell\_is\_marked {

all t: Time - first |

every\_cell\_is\_marked[t.prev] => gameBoard.cells.t = gameBoard.cells.(t.prev)

}

// 4. Players alternate moves.

pred Players\_alternately\_move {

no t: Time - last, t': t.next |

(some LastPlayerToMove.person.t) and

(some LastPlayerToMove.person.t') and

(LastPlayerToMove.person.t = LastPlayerToMove.person.t')

}

// 5. There is no interrupt in the play: If cells are empty at time t-1,

// and there is no winner at time t-1, then there will be one

// fewer empty cells at time t. If there is a winner at time t-1,

// then there will be no change to the number of empty cells at

// time t (per invariant 2).

pred Progressively\_fewer\_empty\_cells {

all t: Time - first |

not every\_cell\_is\_marked[t.prev] and not winner[t.prev] =>

#empty\_cells[t] < #empty\_cells[t.prev]

}

// 6. Player1 marks cells O and Player2 marks cells X.

pred Players\_mark\_cells\_appropriately {

all t: Time - first |

not every\_cell\_is\_marked[t.prev] and not winner[t.prev] =>

let c = gameBoard.cells.t - gameBoard.cells.(t.prev) |

c[Row][Col] = X =>

(LastPlayerToMove.person.t = Player2)

else

(LastPlayerToMove.person.t = Player1)

}

// 7. When there is a winner or when all cells are marked,

// then the recording of "last player to move" is blank.

pred LastPlayerToMove\_remains\_unchanged\_after\_win\_or\_all\_cells\_marked {

all t: Time - first |

((every\_cell\_is\_marked[t.prev]) or (winner[t.prev])) =>

no LastPlayerToMove.person.t

}

// This provides one place that you can call to

// have all the constraints enforced.

pred game\_is\_constrained\_by\_these\_constraints {

Each\_cell\_has\_at\_most\_one\_mark

gameBoard\_remains\_unchanged\_after\_win

gameBoard\_remains\_unchanged\_after\_every\_cell\_is\_marked

Players\_alternately\_move

Progressively\_fewer\_empty\_cells

Players\_mark\_cells\_appropriately

LastPlayerToMove\_remains\_unchanged\_after\_win\_or\_all\_cells\_marked

}

// Return the set of empty cells at time t.

// This is implemented using set subtraction.

// (Row -> Col) is the set of all possible combinations

// of row and col. Subtract from that the set

// of (row, col) pairs containing a mark at time t.

fun empty\_cells[t: Time]: Row -> Col {

(Row -> Col) - gameBoard.cells.t.Mark

}

// Once the game board is completely marked,

// there won't be a "last player." Ditto for when

// there is a winner. That's why there "may" be

// a last player at time t. That is, there isn’t

// necessarily a player involved at every time step,

// i.e., there isn’t necessarily a (Player, Time) pair

// for every value of Time.

one sig LastPlayerToMove {

person: Player lone -> Time

}

// Return the mark (X or O) on board[r][c] at time t,

// or none if there is no mark.

fun board [r: Row, c: Col, t: Time]: lone Mark {

gameBoard.cells[r][c].t

}

// There is a winner when (a) there is a row

// with all X's or all O's, or (b) there is a col

// with all X's or all O's, or (c) there is a left-to-right

// diagonal with all X's or all O's, or (d) there is a

// right-to-left diagonal with all X's or all O's.

pred winner [t: Time] {

some m: Mark |

some r: Row | all c: Col | board[r, c, t] = m

or

some c: Col| all r: Row | board[r, c, t] = m

or

board[first, first, t] = m and

board[first.next, first.next, t] = m and

board[first.next.next, first.next.next, t] = m

or

board[last,last, t] = m and

board[last.prev, last.prev, t] = m and

board[last.prev.prev,last.prev.prev, t] = m

}

// Every call of the game board is marked when

// the set of cells with marks equals all combinations

// of (row, col)

pred every\_cell\_is\_marked[t: Time] {

gameBoard.cells.t.Mark = (Row -> Col)

}

// Initially the game board has no cells.

// One of the players is first to play.

// The game is constrained by the invariants.

pred init [t: Time] {

no gameBoard.cells.t

one p: Player | LastPlayerToMove.person.t = p

game\_is\_constrained\_by\_these\_constraints

}

pred doNothing [t: Time] {

gameBoard.cells.t = gameBoard.cells.(t.prev)

}

pred Play {

init[first]

all t: Time - first |

X.marked\_on\_gameboard\_at\_time[t]

or O.marked\_on\_gameboard\_at\_time[t]

or doNothing[t]

}

pred marked\_on\_gameboard\_at\_time [m: Mark, t: Time] {

some r: Row, c: Col {

gameBoard.cells.t = gameBoard.cells.(t.prev) +

{r': Row, c': Col, m': Mark | r' = r and c' = c and m' = m}

}

}

run Play for 3 but 12 Time