Dade Wood
daw1882
11/26/21

Combinatorial Tron

Motivation:

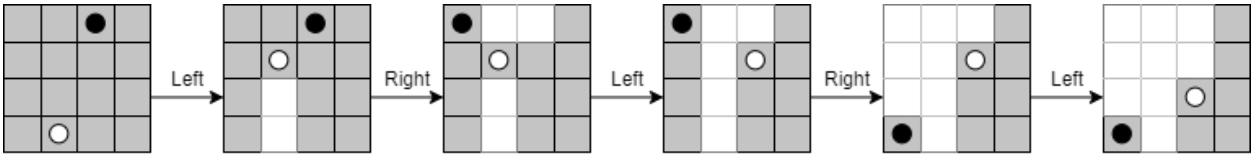
Combinatorial Tron is a two-player game based off of the bike scene in the movie Tron. However, instead of the bikes crashing into walls for an ending condition as in the movie, the last person able to move one of their Tron bikes is the one who wins. What is interesting about this game is it is a turn-based version of what is meant to be more of a reaction time game, and even though the rules are fairly simple, keeping track of how many spaces one has left to move in can become quite complex as the board size and number of bikes increases.

Game Description:

Position: A rectangular grid of unit squares with possibly some squares removed (which cannot be played upon). Each square either occupied by at most one piece called a *tron bike* (either black or white) or unoccupied.

Moves: A player's tron bike moves like a chess rook – any number of squares in a horizontal or vertical line (no diagonals). After the tron bike makes its move, every square it passed through is destroyed. A tron bike cannot cross or enter a destroyed square or a square occupied by a tron bike of either color.

Sample Game: A portion of a game on a 4 x 4 with a single tron bike for each player with Left (white) moving first is below. Left wins after 5 moves because there are no more legal moves remaining for Right.



Game Comparison:

Amazons is one of the most similar games from the ones we have learned this semester. Both Tron and Amazons have ways to destroy squares and allow pieces to move long distances, but in Tron the pieces move like Rooks instead of Queens and they do not have any additional parts like throwing a spear – they destroy squares simply by moving.

Another similar game is Bridg-It since both games involve creating paths that cannot be crossed by the other player while trying to leave spaces open to play for yourself. Unlike Bridg-It, though, the goal of Tron is to be the last person to move, not to simply get to the other side. Tron also has pieces to move as well instead of simply drawing the path and allows for a longer line to be created due to having the same movement style as a chess Rook.

Analysis:

To analyze this game, I have created a helper program to represent a single Tron position as well as a backtracking algorithm to find the outcome class for a position. The outcome class is found by going through the tree of possible moves for a position and recursively assigning the class to each sub-position until the final outcome class is decided. Also, in order to simplify the analysis and allow a game to be analyzed within a reasonable time, I only use 1 Tron bike for each player and do not use a board size larger than a 4x4. These two changes help to limit the number of possible moves a starting position will have and thus allows the program to be able to finish in a realistic time frame since in order to find an outcome class it needs to explore most of the moves for each player. (Note: I say most because it is possible to prune a branch in the game tree and find an outcome class faster than you would by checking every single possibility.) Below is a table with the results from using this program to find the outcome classes of different board sizes with various starting locations for the Tron bikes.

To start, we will examine a simple board size of 1xn:

1) 1x2 board: This is a trivial outcome class of P since there are only two squares and each one contains a Tron bike already. Neither player can move at the start so whoever goes first will lose.
2) 1x3 board: Let Left's bike be located on the leftmost square and Right's bike be located on the rightmost square. This position has an N outcome class as whoever moves first is able to take the square in the center and will win the game. Now let Left's bike be located in the center square and Right's bike be located on the rightmost square. This position will have an outcome class of L because Left's bike is the only one with access to an open square. The opposite would occur and the outcome class would be R if you were to switch the bikes' locations.
3) 1xn board:
    a. Let Left's bike be located on the leftmost square and Right's bike be located on the rightmost square. The outcome class will be N as whoever goes first is always able to

move across the entire row to the opposite end and block the other player from moving.

b. If only one player is on an edge space while the other is anywhere on the board, then the player who is not starting on an edge space will win. This is because the player not on an edge space will always be able to either move to the opposite end or move to the other player's piece and block them from moving.

c. If neither player is starting on an edge space, then the first player will win if they have a larger distance from the closest edge then the second player's distance from either the edge or to the first players bike, otherwise the second player will win. If the first player moves towards the second player's bike they will automatically lose since the second player can simply move the remaining squares between them or can continue moving towards the edge. If the first player moves towards the edge then they can only win if they have more spaces to the edge than the second player does to either the edge or the first players bike since the second player can choose to go either direction on his turn.

Next, we will examine 2xn games (n > 1):

1) 2x2 board: No matter what the starting positions for the players' bikes, the second player to move will always win, as shown in the table below. This is because on each players' turn they will only have one space to move to so after the second player's turn there will be no more valid spaces.
2) 2x3 board:
    a. N positions: From the set up in the table below, these positions are N because whoever is able to get to row 1, column 2 blocks the other player from the remaining spaces and wins the game.
    b. R/L positions: These are the positions such that two players are adjacent in a row to start. Whoever's bike is in the center column will always win because they block off the center whenever it is their turn and always be able to win.
    c. P positions: Every remaining starting position for the 2x3 board. For all these positions, the second player will win every time because they can use the property of symmetry and copy the first player's move until there are no remaining spaces.
3) 2xn boards: As can be seen from the table, the outcome of a position depends heavily on the starting locations for a bike as well as if n is even or odd. In situations where the bikes are places symmetrically on an even board, the second player will always be able to win by

mirroring moves. When the players are adjacent at the top or bottom of the board, the player who is furthest from an east or west edge will always be able to win since they can block the opposing player into the smaller section of the board. Left or Right can also always win in cases that they are able to trap the opposing player on their turn. This happens in two cases in the 2x4 part of the table below and will happen in similar situations to those. N positions are the cases where, depending on who goes first, the other player will be trapped in a symmetrical move loop or will be blocked in some way by the first players initial move and lose the game from the start.

As we continue to scale up in board size, we can notice some common patterns for this game that will continue to apply for all games. First, for any even board set up with two Tron bikes in symmetrical locations, the second player will always win by using a mirroring strategy. Second, the first player will always win in cases where they are able to block the second player into a piece of a board that is smaller or corner them into an area where they will run out of pieces. This strategy is dependent on the bikes being set up in a nonsymmetrical way and in a manner that allows the first player to make a move that separates the board into a smaller section and larger section. Finally, L and R outcome classes occur when one player begins in a situation in the board that they will lose no matter what due to beginning in a space that either begins blocked by the other player or is able to be blocked by the opposing player on their next move.

Conclusion:

Tron is a much more complex game than I first thought it was going to be when starting my analysis. If I had not limited the board size and number of bikes allowed, the game trees would have become very large very quickly. Even with the limitations I set, the best my program was able to do within a reasonable time is a 4x4 because the bikes are able to move as many spaces as they want in a horizontal or vertical direction as long as there is nothing in the way which leads to many possible positions from a single bike being moved. A good thing that came from this realization, however, is that it led me to thinking of many possible variations to the rules that could make the game more interesting/challenging for players. For example, a variation could be to restrict the bikes to being able to move only one square at a time but also include diagonals, restricting the number of bikes allowed on the board, or only allowing the bikes to start in specific locations.

Overall, I found this game to be interesting to analyze and find patterns in. I also enjoyed creating the program to find outcome classes for the starting positions, but I did find it hard to improve efficiency in

the game since there were so many possible moves for any one position. I think for a future project, it would be interesting to further improve this program to be able to handle larger boards and fully be able to test different patterns that I have seen so far.

Outcome Class Table

Below is a table containing outcome classes found by my program for different starting positions. The board size is formatted as row by column and the bike coordinates are formatted as (row, column). Runtime is given in seconds. For instructions on running the code, view documentation in .py files.

Board Size	Left Bikes	Right Bikes	Outcome Class	Runtime
1x2	(1, 1)	(1, 2)	P	<0.01s
1x3	(1, 1)	(1, 3)	N	<0.01s
	(1, 1)	(1, 2)	R	<0.01s
1xn	(1, 1)	(1, n)	N	
2x2	(1, 1)	(2, 2)	P	<0.01s
	(1, 1)	(1, 2)	P	<0.01s
2x3	(1, 2)	(2, 2)	P	<0.01s
	(1, 1)	(2, 1)	P	0.16s
	(1, 1)	(2, 3)	P	0.03s
	(1, 1)	(1, 2)	R	0.03s
	(1, 1)	(1, 3)	N	<0.01s
	(1, 1)	(2, 2)	N	0.02s
2x4	(1, 2)	(2, 3)	P	0.06s
	(1, 2)	(1, 3)	P	0.16s
	(1, 1)	(2, 4)	P	0.12s
	(1, 1)	(1, 4)	N	0.03s
	(1, 1)	(2, 3)	L	0.06s
	(1, 1)	(1, 3)	R	0.11s
3x3	(1, 2)	(3, 2)	N	0.20s
	(1, 1)	(3, 3)	N	0.34s
	(1, 1)	(2, 2)	P	0.53s
	(2, 1)	(2, 2)	R	0.14s
	(2, 1)	(3, 2)	N	0.06s
	(2, 1)	(3, 3)	L	0.11s

3x4	(1, 2)	(3, 3)	P	3.20s
	(1, 1)	(3, 4)	N	10.61s
	(1, 1)	(1, 4)	N	3.16s
	(1, 1)	(1, 3)	R	4.52s
	(1, 1)	(1, 2)	R	4.00s
	(1, 1)	(2, 1)	R	2.31s
	(1, 1)	(3, 1)	N	4.22s
	(1, 1)	(2, 2)	R	3.44s
	(1, 1)	(2, 3)	R	9.55s
	(1, 1)	(2, 4)	R	1.70s
	(1, 1)	(3, 2)	N	2.47s
	(1, 1)	(3, 3)	P	5.87s
	(2, 2)	(2, 3)	P	4.71s
	(1, 2)	(1, 3)	P	4.25s
	(1, 2)	(3, 2)	N	1.23s
4x4	(1, 2)	(4, 3)	P	214.46s
	(1, 1)	(4, 4)	P	427.49s
	(2, 2)	(3, 3)	P	266.13s
	(2, 2)	(2, 3)	N	342.75s
	(2, 1)	(2, 4)	N	10.36s
	(2, 1)	(2, 3)	N	97.26s
	(2, 1)	(2, 2)	R	68.03s
	(2, 1)	(3, 3)	L	295.64s
	(2, 1)	(3, 2)	R	66.41s
	(2, 1)	(3, 1)	P	94.66s