TRON Game Simulator

User Guide

1. Introduction

Tron is a game where 2 players compete with each other in a Snake-like gameplay. Each player moves with equal speed and leaves behind them a non-dissapearing trail which acts as an obstacle that neither of the players can cross. Objective of the game is to manoeuver opponent into hitting one of the trails or drive off the map, while simultaneously avoid doing the same. This simulator enables two bots (written in python or javascript) to compete on a configurable map.

2. Game map

Bots compete with each other on 2D plane. Game map has configurable width and height, as well as possibility of defining obstacles. Map of size 30x30 has 30 rows and 30 columns of cells. Each cell stores one value which defines the state of the cell:

- 0 if cell is empty
- 1 or 2 if bot, or bot's trail occupies the cell. At the beginning of the game each bot is assigned a number with which it will mark visited cells.
- 8 if cell is beyond map (more in gameplay section)
- 9 if obstacle occupies the cell

example map could look like this:

0	2	0	0	0	0	0	0	0	0
0	2	2	2	2	2	2	2	0	0
0	1	1	1	0	0	0	2	0	0
0	1	0	1	0	0	0	2	0	0
0	1	0	1	9	9	0	2	2	0
0	1	0	1	9	9	0	0	2	0
0	1	0	1	1	1	0	0	2	0
0	1	0	0	0	0	0	0	2	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Each cell has coordinates (row, column) starting with left top corner:

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

2. Game Setup

When starting a game you can define height and width of a map, or create map with a configuration file. When map configuration file is inputted it overrides hand-chosen width and height.

Configuration file should be a text file with format:

{height}x{width}
{obstacle0_start_coordinates}-{obstacle0_end_coordinates}
.
.
.
{obstacleN_start_coordinates}-{obstacleN_end_coordinates}

example configuration file contents:

40x50
8,5-30,5
10,4-10,18

This configuration will create map with 50 width, 40 height and 2 straight line obstacles. One starting at (8,5) and ending at (30,5); second starting at (10,4) and ending at (10,18).

Obstacles defined in configuration file have to be either single point (e.g. 4,6-4,6) or straight lines. If defined obstacle has other form (e.g 4,6-9,2) simulator will ignore it.
Obstacles should be created **left-to-right** and **top-to-bottom** as ilustrated by example (coordinate that is changing is “rising” between start and end)

Additionally bots do not see all map, but a snippet with configurable size. You can configure front and side size. Front size=5 and side size=2 will result in bot seeing a snippet that is 5 cells long and 5 cells wide (2 cells on each side + cells directly ahead)

3.Gameplay and communication

Communication between bots and simulator is accomplished using Standard input and output. This leads to maximum compatibility in tools with which bot can be written.

When bot is spawned, It's Id number ("1" or "2") is delivered to it by simulator as command line argument. e.g. **python home/user/bot.py 1**

Making a move by bot is accomplished with the following algorythm:

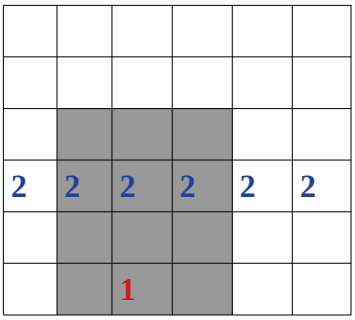
1. Simulator sends bot snippet of the map

using following format:

"map---row.column.value:row.column.value: …… :width"

with each cell being described by row, column and value separated with dot (".”). And each cell separated with colon (":"). At the end is added width which is a number describing how wide the view is (for side view=2 this parameter would be 5).

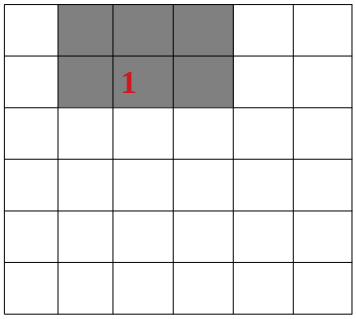
For front_view=3 and side_view=1, map piece for bot 1 would present as follows:



And string sent to bot would be:

"map---**2.1.0**:**2.2.0**:**2.3.0**:**3.1.2**:**3.2.2**:**3.3.2**:**4.1.0**:**4.2.0**:**4.3.0**:**5.1.0**:**5.2.1**:**5.3.0**:3"

in case the bot is looking “beyond” map, “phantom coordinates” are calculated and given value “8”



Here bot looks 2 cells forward beyond map. Sent map piece would look:

(-2.1.8) (-2.2.8) (-2.3.8)
(-1.1.8) (-1.2.8) (-1.3.8)
(0.1.0) (0.2.0) (0.3.0)
(1.1.0) (1.2.1) (1.3.0)

“phantom points” written in grey colour.

Depending on direction in which bot is currently headed map piece cells are send in string in orders depicted below:

			0	1	2			
			3	4	5			
			6	7	...			
2	5	...		^		6	3	0
1	4	7	<	B	>	7	4	1
0	3	6		v		...	5	2
			6	7	...			
			3	4	5			
			0	1	2			

After receiving map piece bot should respond with “MAP_RECEIVED”

1. Bot sends next move to simulator

Simulator will send prompting message to bot -“move”.

After receiving this message bot should respond with “MOVE---row---column” containing row and column of it’s next move.

It’s up to bot to send correct coordinates, if simulator receives coordinates which translate to invalid move, like moving across or more than one cell, It will discard the move, and bot will not move in this turn.