Scala project

Tron bot challenge

Author :
Arthur Douillard

Teacher :
Adrien Broussolle

Février 2017

1 Functional programming

The first bot was supposed to be done using the functional style of Scala. While functional programming is pretty, it's not very adapted to challenge like this, where mutable data structures and loop the best choice.

For the functional bot, I assumed it'll only face one opponent.

1.1 The grid

At first I went for a 1D-Array of Int to represent the map. It was initialized at '-1'. When a lightcycle was going on a tile, the tile was adopting its id value. I also wrote some methods to interact with the 1D-Array as it was a 2D-Array.

Because I didn't like very much to have value hard-coded at several places (like the '-1' value), I switched from Int to a trait : The sealed trait State, used by the case classes : Empty() and Occupied(id : Int).

1.2 The A.I.

1.2.1 Unranked

At first I've written a bot that was making random moves (but valid). Not very efficient as you may guess.

1.2.2 Wood league

By imposing an order to the moves priority (UP -> LEFT -> DOWN -> RIGHT) the bot started making "snake" moves.

It's not the smartest bot but it still got the top (between first and tenth) of the wood league.

1.2.3 Silver league

Then I've coded some kind of minmax, where each node (representing a moment of the game in the future) has 4 sons for the 4 possible directions.

If at any moment during the tree traversal I stumbled upon an invalid tile or a tube-shape zone (zone where only one move is possible) the branch is pruned immediately by returning a very low score.

When reaching a new node, I :

— Change the state of the tile I'm currently on to Occupied.

— Create 4 branches (for the 4 directions) and go recursively through them. Each branch receive a copy of the grid (not very efficient but using a queue to stash the modifications was barely better).

If the depth reach a defined max depth (usually 8 in order to finish the move in less than 100ms) I'm returning the score of my current position. The score is computed by looking at the neighbors of the tile we arrive to in the end.

This algo leads me to the Silver League.

2 Imperative

For the second bot, almost everything have been rewrited.

2.1 Grid

This grid was almost as the first one. I used a 1D-array of State. Three case classes were existing :

— Empty()

— Occupied(id : Int)

— Predicted(id : Int)

The last case class is used in the Voronoï to make the difference between the reachable tiles and the real tiles.

2.2 The A.I.

2.2.1 Voronoï

I wrote an implementation of the flood fill algorithm that, at the end of its execution, results in a Voronoï diagram.

I launch a small 'tree' of 4 leafs, for the four directions (minus the invalid direction). For each direction I used the flood fill algo :

Basically each player expand its latest positions (yes plural) in all the possible directions turn by turn until there are no more reachable tile.

Then I'll count the number of tiles claimed by me, times an arbitrary coefficient minus the number of tiles claimed by my opponents.

Thus for each directions I'll get a integer score, I'm choosing the maximum score.

This naive implementation leaded me to the top Silver league.

2.2.2 Different minor tweaks

One tweak that really helped me is that to change the order in which each player play. Since I was going in a direction at first, it means I've played before the others. Then the order switch from (0 -> nbPlayers) to ((me + 1) -> nbPlayers -> 0 -> me).

This leaded me to the Gold league.

Then when several directions had the same score, I've decided to implement a wall hugger : I'd choose the direction keeping me next to a wall. This helped me fill more effectively closed chamber.

I've also add a lookup of 2 : I would choose two consecutive directions, let my enemies flood two turns, and then execute the classic voronoi with everyone (me included) flooding.

This leaded me to the Legend league, in a quite good position (100-120).

2.2.3 Minmax

I've tried to implement several versions of my Minmax, some were pruning too much and gave bad results, others were taking too much time (even with only one depth level).

If I had more time (I'm also a Teacher Assistant) I would have tried to implement a Minmax where I'd prune all players that are either too far or in a closed chamber. I'd also switch my board game structure from case classes to integer.