Intelligent Systems 2014

Planet Wars Bots



**Authors**

Anouk Boukema

Baudouin Duthoit

Gossa Lô

# Introduction

Blablabla

# Bot #1 : FirstBot

## Pseudo code :

DoTurn():

Source = FindThePlanetWithTheBiggestFleetWeOwn();

PossiblesDestinations = InterestingPlanets();

Destination = FindTheFirstCapturablePlanetOf(PossiblesDestinations);

IssueOrder(Source, Destination);

InterestingPlanets():

List;

for (planet in NotMyPlanets):

If (planet.GrowthRate > planet.Fleet):

List += planet;

List += enemyPlanets;

return List;

## Explanations :

This first code we implemented finds the biggest fleet available by scrolling through all planets and comparing their number of ships??. (to find the one with the biggest fleet). This method will be used for most of the algorithms.

To find the destination, we first sort all the available destinations. To do so, we first choose all planets with more Growth Rate than Ships on it (we'll get paid back thus), then we'll add all the enemy planets, because it's always beneficial to attack the enemy. After that, with this sorted list of planets, we choose the first one that we can capture. This bot is especially efficient in serial mode.

## Why ? :

# Bot #2 : HillclimbingBot

## Pseudo code :

DoTurn():

source, dest, D=-100;

For (s in possibleSources)

For (d in possibleDestinations)

If (DCalculation(s,d)>D)

source = s;

dest = d;

D = DCalculation(s,d);

IssueOrder(source, destination);

DCalculation(source, dest):

return MyShipsLoss - EnemyshipsLoss + MyGrowthRateImprove - HisGrowthRateImprove

## Explanations :

This code is just finding the best attack to do depending on our heuristic function : DCalculation(). For this, we just go through all couple of source/destination and calculate the heuristic value of it.

In DCalculation, we only count the difference between the current state and the next one, by comparing, the loss of ships and if the growth rate is increasing for us and/or decreasing for the enemy.

## Why ? :

# Bot #3 : BeamsearchBot

## Pseudo code :

DoTurn(pw):

root = Node(Simulation(pw));

Beam = Array<Node>(3); // Will contain our current open nodes

Beam.add(root);

While (remainsTime):

For (node in Beam): // Don't stop looping on Beam until the time is gone (1)

If (node.isALeaf()):

break;

Else:

For (son in node.Sons()): // (2)

If(Sons.value()>min(Beam))

Beam.del(min(Beam));

Beam.add(son);

Beam.del(node);

IssueOrder(max(Beam).Source, max(Beam).Destination);

Node: Class :

SimulatedPlanetWars

Source // the source used in the first step to get there)

Dest // the destination to get to the that state)

Value // the heuristic value)

Depth // in case of an A algorithm, not used)

Sons() // Function that return all the possibilities from that point

// WARNING : Sons() create possibilities by simulating a FirsBot attack !

## Explanations :

Beam search is an algorithm of research where we only store the N best options (here n=3). So, here, we store the options in "Beam" of size N. First, while there is still enough time, we loop on this array (1). Then we loop (2) to find the sons for each node of this array and if one of the sons is better than the N first solution, the son will replace the weakest solution, then the next son will be compared to this minimum too.

When we run out of time, we just send the best current solution. The value is calculated on a DCalculation seen previously.

## Why ? :

# Bot #4 : FirsParallelBot

## Pseudo code :

DoTurn():

Source = FindThePlanetWithTheBiggestFleetWeOwn();

PossiblesDestinations = InterestingPlanets();

Destination = FindTheFirstCapturablePlanetOf(PossiblesDestinations);

IssueOrder(Source, Destination);

InterestingPlanets():

List;

for (planet in NotMyPlanets):

If (planet.GrowthRate\*2 > planet.Fleet): // 2 growth before an ennemy reaction

List += planet;

List += enemyPlanets;

return List;

## Explanations :

This Bot is essentially identical to firstBot except that more planets are declared as interesting.

This first parallel code we implemented find the biggest fleet available by scrolling through all our planets to find the one with the biggest fleet.

To find the destination, we first sort all the available destinations. To do so, we first choose all planets with 2 times more Growth Rate than Ships on it (we'll get paid back thus), then we'll add all the enemy planets, because it's always beneficial to attack the enemy. Then, with this sorted list of planets, we choose the first one that we can capture. This bot is especially efficient in serial mode.

## Why ? :

# Bot #5 : MyAdaptiveBot

## Pseudo code :

DoTurn():

readLearningValueFromFile()

Attack(type):

source = OwnBiggestFleet();

Switch(type):

Case : Planets

dest = FarestCapturablePlanet()

Case : GrowthRate

dest = BiggestCapturablePlanet()

Case : Ships

dest = StrongestEnnemyPlanet()

Case : Dcalculation

dest = max of DCalculation + Distance in availableDestinations()

Defend():

reinforceTheBiggestPlanet();

## Explanations :

The main idea was to build a decision tree able to learn by himself (go more on the right, or more on the left by adding some learned values to the tests). Each node contains a test to decide if it goes on the right or on the left. If there is a hesitation or an equality, we choose randomly one side to be less predictable, all the tests are comparisons with the learned value which are often zero. We were running out of time to implement properly the learning algorithm, so we only implement a reading function inside a file to initiate the learning values. The update function is able to write inside the file, but what to write is another issue. We currently know how to write at the end of each turn but not at the end of a game, that would be our next step for improving our algorithms.

## Why ? :

# Conclusion