Intelligent Systems 2013

Planet Wars Bots



**Authors**

Anouk Boukema

Baudouin Duthoit

Gossa Lô

# Abstract

of 2-3 paragraphs

# Introduction

Introduction (to the problem, but also your solution, en some results. 2 pages)

# Background information

Background information: description of the game, the challenge, the IS framework, whatever is necessary to understand your paper. Here you would normally also summarise related work, but this is not required here as all the methods are in the textbook (1-2 pages)

## The game

In the Fall of 2010 there has been a google AI challenge. This challenge (probably) inspired some teachers at the VU. They might have thought that if they would simplify the game given by the contest they could change it into a perfect exercise for second years students of informatic related studies. And so the Planet Wars which is used for this for this report, came into existence.

The creators of the Google challenge based their game on another game called Galcon which has been created by Phil Hassey. Indirectly the game Planet Wars used for this research is also based on Galcon, therefore there will follow some further information about Galcon.

### Galcon

As described on the Galcon site: “Galcon is an awesome high-paced multi-player galactic action-strategy game. You send swarms of ships from planet to planet to take over the galaxy.”

The gameplay for this game is fairly easy. All the options are manageable by mouse. Selections of one or more of the owned can be made. The selected planets will attack or reinforce another planet. By scrolling the percentage of the ships which will create a fleet and attack can be changed. For example when 100 ships are owned and 60 ships are needed to concur an enemy planet the percentage should change to 70, 80 or 90. By doing so a fleet of 70, 80 or 90 ships will fly out to concur the enemy planet. Changing the percentage can be done at any time and as often as the player likes. Galcon is not played in turns and therefore attacks can be made at any moment. Also, planets which are possessed will gain the player more ships over time. The bigger the planet, the higher the time rate the planet will give you a ship.

The goal of the game is to erase your enemy from the map.

### The simplification: Planet Wars

For the IA competition and for this course simplification of Galcon were made. The major change is that there isn’t any variation possible in the percentage of ships send from a planet. This percentage is set on 50.  
The other adaptation is that Planet Wars plays in turns. Bigger planets still give more ships each turn, although this now doesn’t depend on the creation rate of the ships but of the growth rate of a planet. So each turn a possessed planet will generate a fixed number of new ships (dependent on the growth rate number of that planet).   
There are two possibilities when playing in turns.

#### Version 1 serial

Here each turn equals one move of a player. This means the other player can see the move his enemy made and adapt to that.

#### Version 2 parallel

Now, for one turn both players have to make their moves. This means that they won’t know which planet the other player chooses to attack. If both of the players choose the same planet to attack, they will start at the same time, whom will win the planet is now dependent of the length from the source planet to the destination planet. The one that arrives first will concur it, but the one that arrives second might take it over again.

## The challenge

For this course at least four bots had to be created. Each with different specialities. Two of them have to employ state space search. This means that they base their decisions on the observations they made from their environment correct?. The environment in this case is the map, and the possible observations are: the planets with their place on the map, the number of ships they contain, their growth rate and by whom they are possessed (one of the bots or neutral)Or is there more to be observed??.

The other bot had to be adaptive. So it has to be able to change his “tactics” regarding the different enemies and maps. This means that the bot, or might learn to see the best heuristic values from the previous games, or learn the opponents strategy. more??

For the last bot we were allowed to choose other techniques, for example the once we learned at the lectures.

## The IS framework

The environment the bot has to be operating in has a few properties which will be discussed.

The environment of Planet Wars is fully observable for the serial version of the game. The parallel version is partial observable because it the bot is not able to observe all the data needed to make the best choice.

Since there are no other random factors than the move of the enemy the environment is deterministic. Also the game is episodic because it plays in turns and therefore can’t change while the agent is making his decision. Neither is it semi dynamic because

Would this be the exact information about the game itself. So in what kind of framework our bots are working in?

Deterministic, single player,

## More???

# Research questions

Research question: Explain what you did, and what possible outcomes of your setup and contribution. That could e.g. be that you want to find out whether one methods works, or that it works better than other.  What do you mean by works better: Wins once, wins all the time, wins mostly, not alway loose, works faster, works better when time is restricted etc. (1 page)

Is a non-deterministic bot necessary to win planet wars?

Does the efficiency of a bot depend on the maps or the number of planets ?

Can we make a bot that wins a game without passing the time limit once?

Is there a difference between being the starting player compared to being second regarding the winning statistics?

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Wins 12/12 | tie | Crashes | Passed time limit | Ratio of planets concurred | Number of moves | Winning time 2/2: RandomBot | | Winning time 2/2: BullyBot | | Winning time 2/2: LookaheadBot | |
| Start | Second | Start | Second | Start | second |
| FirstBot |  |  |  |  |  |  |  |  |  |  |  |  |
| Hillclimbing |  |  |  |  |  |  |  |  |  |  |  |  |
| Beamsearch |  |  |  |  |  |  |  |  |  |  |  |  |
| Adaptive |  |  |  |  |  |  |  |  |  |  |  |  |

Maybe better explain the table and why we’ve chosen for these.

# Experimental setup

Four different bots which can play the game planet bots are made. Each of them is specialised in a different way. These specialities are based on theories given in the lectures of this course. For each bot their foundation principles will be explained, and the way this principle is implemented in this bot. At the and the four different methods used for the bots will be compared.

Experimental setup: Explain how you set up your experiments. What did you do, e.g. in terms of implementation (brief), but mostly in order to compare your different methods. Define your metrics. (2 pages)

## FirstBot

At the start of this course there was the idea to first make a bot which wasn´t based on the exercises. Just to experiment. That is how the FirstBot was created.

### Implementation

FirstBot is a fairly easy bot which attacks enemy planets with his biggest fleet. He prefers attacking enemy planets he can concur, if not he just attacks the enemy planet which is highest in is list. To experiment a bit more something else was implemented. FirstBot should consider if there aren´t any better options than attacking the enemy. This means FirstBot will gain more ships when he attacks a planet than it had cost him to concur the planet. We concluded this was only possible when the growth rate of a planet was bigger than the ships it possesses.

## HillclimbingBot

The basics of this bot are based on the hill climbing search principle. For this search method the agent compares all the heuristic values that can be observed by him and chooses the best option. By doing so it made a decision from which it can’t get back, it therefore only explores one branch of the possibility tree (is that the real name).

Arguments why hill climbing is a good option, why we’ve chosen it.

### Implementation

To implement this search algorithm in a bot an heuristic values D was created. D indicates the difference between the ships the bot possesses and the ships the enemy possesses. This D may differ for each possible planet the bot can attack.

D planetA  = HissLossA – MyLossA + MyGrowthA - HisGrowthA

This formula indicates that the bot will favour to attack a planet of the enemy because this generates

a high D rate. If the bot doesn’t attack one of the enemies ships the D will most possibly be a negative value. A positive value is only created when the growth rate of a planet is higher than the number of ships it houses, since this is almost (are there maps were this is the case??) never the case the D will be negative.

All the possible D values the bot can find differ over the number of planets in the game and the number of planets the bot possesses. Since 27 planets is the maximum, the most D values the bot possibly has to create are 14\*13 =182.

Our bot will attack the planet that, in combination with (one of) his planets made the highest D value.

## BeamsearchBot

The basic of this bot is based on the beamsearch principle. For this search method the agent compares all the possible heuristics he can observe and chooses the best option. This decision might have given him more observable heuristic values. These new values are now part of the total optional heuristic values the agent can choose from. This means that, if one of the heuristic values from the previous decision seems to be a better option, the agent will choose this node. As result the agent can jump from node to node regarding the best option. Compared to a hill climbing agent, the beam search agent does remembers the nodes he didn´t choose in the previous step. This gives more options and makes it possible to try different paths. For a large tree with high branching rate this can give many optional nodes. To prevent an overload of information the beam search agent only remembers a certain amount of options. When the agent find new options it compares each of them with the rest of options and erases the worst, this might be the new node but can also be one of the previous values stored in the memory of the agent.

Arguments why beam search is a good option, why we’ve chosen it.

### Implementation

To make the beam search principle a possible option for a Planet Wars bot some changes had to be made concerning the step back possibility. In planet wars it is not possible to re do a turn. This means that when there is the need to choose for one of the options from the previous turn, there has to be made a simulation of the game. When this is done values can be found and compared, when one of the previous option was better the simulation restarts with the values from the previous simulation in memory.

## Comparing the methods

# Results

Results: describe your results in some kind of overview tables, and point the reader to the most significant and interesting results in a short text. (2 pages)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Wins 12/12 | tie | Crashes | Passed time limit | Ratio of planets concurred | Number of moves | Winning time 2/2: FirstBot | | Winning time 2/2: HillclimbingBot | | Winning time 2/2: BeamsearchBot | | Winning time 2/2: AdaptiveBot | |
| FirstBot |  |  |  |  |  |  | - | - |  |  |  |  |  |  |
| Hillclimbing |  |  |  |  |  |  |  |  | - | - |  |  |  |  |
| Beamsearch |  |  |  |  |  |  |  |  |  |  | - | - |  |  |
| Adaptive |  |  |  |  |  |  |  |  |  |  |  |  | - | - |

# Findings

Findings: As a separate step interpret the results, and give explanations for the results. (1 page)

# Conclusion

Conclusions: summarise what you did, and highlight the most inportant findings. (1 page)

The paper should describe the chosen methods and compare them analytically and emprically. Based on this analysis you should draw some generic conclusions. For this you should take (at least) the 4 bots you implemented and compare their performance by having them systematically play against each other (not against Random, Lookahead and Bully). According to the different environments (but possibly not) different bots might outperform others. Define some interesting hypotheses and research questions, and use your analysis to verify or falsify them

1. Title page with title and authors, and an abstract of 2-3 paragraphs
2. Introduction (to the problem, but also your solution, en some results. 2 pages)
3. Background information: description of the game, the challenge, the IS framework, whatever is necessary to understand your paper. Here you would normally also summarise related work, but this is not required here as all the methods are in the textbook (1-2 pages)
4. Research question: Explain what you did, and what possible outcomes of your setup and contribution. That could e.g. be that you want to find out whether one methods works, or that it works better than other.  What do you mean by works better: Wins once, wins all the time, wins mostly, not alway loose, works faster, works better when time is restricted etc. (1 page)
5. Experimental setup: Explain how you set up your experiments. What did you do, e.g. in terms of implementation (brief), but mostly in order to compare your different methods. Define your metrics. (2 pages)
6. Results: describe your results in some kind of overview tables, and point the reader to the most significant and interesting results in a short text. (2 pages)
7. Findings: As a separate step interpret the results, and give explanations for the results. (1 page)
8. Conclusions: summarise what you did, and highlight the most inportant findings. (1 page)

Evaluate everything as structured as possible. Let each bot fight other bots, to test their abilities. This has to be done for each bot we made and in the same systematic way.