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)

In this course we are learning more about the theory and practice Intelligent Systems, systems that perceive reason, learn and act intelligently. The goal of the course is to make four bots that can play PlanetWars. PlanetWars is a game that was based on a popular iPhone and desktop strategy game, called Galcon. The creator of Galcon

# 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 a Google AI challenge took place. This challenge was organized by the University of Waterloo Computer Science Club and sponsored by Google. The goal of this challenge was to create strategic bots that could win the game of Planet Wars. The AI challenge inspired teachers at the VU. By simplifying this game, they found a way to challenge the Intelligence Systems students to implement the theory into a practical format.

The creators of the Google challenge based their game of Planet Wars on another game called Galcon, created by Phil Hassey. To know more about Planet Wars, it is useful to have some information about Galcon first.

### 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 maybe 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, instead, 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 simplified version: Planet Wars

For the AI competition and for this course, a simplified version of Galcon was made. The major difference is that variation in the percentage of ships send from one planet to another is not possible. This standard percentage is set on 50.  
The other adaptation is that whereas in the game Galcon, attacks can be made any moment, Planet Wars plays in turns. Bigger planets still give more ships each turn, but this does not depend on the creation rate of the ships, but on 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

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

#### Version 2 parallel

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. Te possessor of the planet is dependent on the length from the source planet to the destination planet. The player first arriving at the planet 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. The goal of this process is finding a goal state with a desired property. Decisions on the observations they made from their environment correct? HMMM nog ff checken. 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 (by one of the bots or neutral).

The third bot had to be adaptive. This means that it has to be able to change its *tactics* regarding the different enemies and environments. The bot should either learn to be able to observe the best heuristic values from the previous games, or to learn the opponents strategy.

For the last bot, we were allowed to choose other techniques, for example the one we learned at the theoretical 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 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. The game is episodic, since it is played in turns and therefore cannot change while the agent is making its decision.

While the game Galcon can be played both single -and multiplayer, the Planet Wars game used in this course is single player. The bots made in this course will compete against the computer.

DOESN"T THIS MAKE IT A MULTIPLAYER??

## More??? kijk ik nog ff naar later

# 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)

After the creation of four bots they needed to be tested. This will be done on the basis of several research question. Why there has been chosen for these specific questions will be explained in this chapter. In this chapter the way of testing these research questions and which measurements are needed will also be explained, than in chapters “Conclusion” or better also in findings or explanation of the results the answers to the research questions will be given.

*“Does the efficiency of a bot depend on the maps or the number of planets?*”  
(lets please not check each bot for each map that’s crazy)   
Explanation: some humans feel the need to concur the whole world and not just their enemy. The question arose if it is wise to do so? And if so would it also work in the game Planet Wars.   
In revers if it true that when a bot does not only attack the enemy planets but also neutral ones, will it perform better.   
Way of testing: The ratio of neutral planet concurred have to be calculated for each bot and each different planet. This will be done by counting all the neutral planets the bot attacked and dividing this by the total number of neutral planets that could be attack at the start of the game. Or is it more fair to say (I think so) that it should be divided by the number of neutral planets the bot could have possibly taken over. (So if the enemy attacked first he couldn’t have this neutral planet) Which would give

Required measurements:

* number of neutral planets attacked by bot
* number of neutral planets attacked by enemy bot
* number of total neutral planets at the start of the game

*“Is there a bot that wins but never passed a turn?” is this the same as asking which bot is best?*

Explanation: If a bot passed a turn this means it took to much time thinking. Thinking in this case means making calculations about which move would be the best. The more calculations a bot makes the better the result. But then again, when a bot takes too much time calculating he can’t do a move. Many discussions can be made about which option is worse, not making a move, or not making enough calculations. But everyone should agree that making as many calculations as possible and not passing a turn is best.

Way of testing: The assumption is made that a bot which makes as many calculations as possible wins over a bot that does less calculations. With this assumption the test is fairly easy. For each bot that never passed a turn it should not lose the game for which this was true. Of course there is a possibility that one of the bots might even pass this test for every game played, therefore the number of times a bot passes the test will also be measured and against whom (or is total number of passed test enough to find the best bot)

Required measurements:

* number of passed turns for each game
* related wins or losses for each of these games

*“are there bots that are equally good”*

Explanation: since each of the four bots uses different algorithms as basis, they might as well perform very different on the same enemy bots or maps (maybe to much work to test). If they would not perform different, one of them would be superfluous. The useless one of the two should than be the bot that uses more code and time than the other one, since ……………………

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

*A bot made for the parallel type of Planet Wars will always win from the ones made for the serial type?*

1. Is a non-deterministic bot necessary to win planet wars?   
   What’s the meaning of this question. A deterministic bot is a bot that doesn’t contain any randomness. But since the game itself is already deterministic (explained in the IS framework), why would the bot not be deterministic and therefore why would you need such a non-determenistic bot to win

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 Wars 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. And finally, 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 make a bot which was not based on the exercises, to experiment. That is how the FirstBot was created.

### Implementation

FirstBot is a fairly easy bot which attacks enemy planets with its biggest fleet. It prefers attacking enemy planets it can concur, if not, it attacks the enemy planet which is highest in its list. To experiment a bit more, something else was implemented. FirstBot should consider if there are not better options than attacking the enemy. This means FirstBot will gain more ships when it attacks a planet than it would have cost to concur the planet. We concluded that 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 and chooses the best option. By doing so, it makes a decision from which it cannot go back, it therefore only explores one branch of the possibility tree (is that the real name). G: Yeah, I think so

The reason that Hill Climbing is a good method in trying to win Planet Wars is because it is a relatively simple algorithm. This algorithm is widely used when thinking time in a game is limited.

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. The 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 does not attack one of the ships of the enemy, 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 differs 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 its planets made the highest D value.

## BeamsearchBot

This bot is based on the beam search principle. For this search method, the agent compares all the possible heuristics it can observe and chooses the best option. This decision might have given it 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 a result the agent can jump from node to node observing the best option. Compared to a hill climbing agent, the beam search agent does remembers the nodes it did not choose in the previous step. This gives more options and makes it possible to try different paths. For a large tree with a 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 finds new options, it compares each of them with the rest of the 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 redo a turn. When there is the need to choose for one of the options from the previous turn, a simulation of the game has to be made. When this is done, values can be found and compared. If 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)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Map 1 | 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 |  |  |  |  |  |  |  |  |  |  |  |  | - | - |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Map 2 | 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.