voorpagina

# Introduction

## Artificial intelligence

The online encyclopedia gives various definitions of the term artificial intelligence.

The encyclopedia searches on the web for multiple sources with an definition and they all combine the words computer technology and intelligence. [[1]](#footnote-1)

There is one particular definition that we regarded as odd, which was the definition of Wikipedia.

/\* moet nog naar het engels \*/“*Het is moeilijk te definiëren wat 'intelligentie' precies is. Het is derhalve ook moeilijk te definiëren wat artificiële intelligentie precies is.”* [[2]](#footnote-2)

This quote means initially that is quite hard to define intelligence, and therefore is it hard to define artificial intelligence.

Discussion on intelligence and how to perceive intelligence are still undecided.

Some think one is intelligent when one is able to learn, where another perceives intelligence as not only as a matter for brains but also a matter of emotion.

Considering a machine is able to recognize emotion but not experience it, a machine is not able to be intelligent.

A possible test to examine whether a machine is qualified to be calls intelligent is the Turing test.

This test includes that a machine portraits itself as a human and if the human interacting with this machine does not recognize that it is a machine, the machine could be called intelligent.

This trick is a good method to examine whether a machine is intelligent of not.

A counter argument for this is the Chinese room argument.

This argument holds that understanding the output does not mean that the machine also understands the input.

The process from input.

Without entering more in this discussion we define artificial intelligence as technology that is able to operate and react to it's environment, without the influence of humans.

In our research we have tried to create a bot that is able to play a game of Planet Wars without any human influence.

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

# Background information

## Description of the game “Planet Wars”

Planet Wars was inspired by Galcon Fusion, a simple strategy game. The goal is to dominate space and conquer all planets.

You have two players, player Blue and player Red. The idea behind Planet wars is quite simple.

In order to conquer a planet you have to attack the other planets with your ships and destroy their ships.

One of the players Red or Blue  would like to be the dominator of space and they attack each other and neutral planets to gain more power.

Screenshot

To conquer a planet you have to send as many or more ships than are positioned on the planet that you would like to conquer.

The idea is that the ships destroy each other leaving the planet without defense and free to take over.

Note that you cannot decide how many planets you send. You always send half the ships that are available on the planet.

The ships you send will arrive at the end of your turn, so you can also issue one order a turn.

Screenshot

Every planet is also producing ships, in order to gain more ammunition to attack other planets or to defend themselves.

Planets with a large growthrate are able to produce more ships is a small timeslot than planets with a small growthrate.

It is possible to play the game serial and parallel to each other.

We first assume that we play the game serial and both players are waiting for their turn.

Later on in our research we perceive the game in a parallel manner and both players are able to attack at the same time.

## Challenge

The Artificial Intelligence Challenge, better known as the Google AI Challenge is a worldwide challenge started by the University of Waterloo Computer Science Club. [[3]](#footnote-3)

The Computer Science Club of the University of Waterloo organized the challenge for their own students.

So initially only Waterloo University students were allowed to compete.

In 2010 Google started to sponsor the challenge which made it an public event.

Now it's reach is worldwide and students from all over the world compete in the challenge.

The challenge for the game Planet Wars started in September 2010.

The goal was to build a bot that could play other bots. [[4]](#footnote-4)

The most important aspect is that there is no human influence which is a particular AI attribute.

The winner of the AI Challenge 2010 was Gabor Melis.

He surprisingly used Lisp to program his bot, instead of a bot programmed in C++ which most participants of the challenge submitted. [[5]](#footnote-5)

There are quite a few competitions that encourage new approaches for AI problems.

In the AI game division it is key that a program is adaptive and can change its strategy without human intervention.

The bot's that are programmed should also compete against each other, so the programmer is not able to know the game in advance.

This was also our own challenge.

It is quite difficult to program a bot that will win against every other bot.

Since you don't know the strategy of the opponent, it is hard to come up with a defense mechanism.

With this is mind you would like to imagine every possible strategy and possible defense.

If you would implement this is a bot it would take too much time and space to compute the right move.

So we tried to think of a couple strategies that would lead us to the win as often as possible.

This will be discussed more elaborate in our hypothesis.

## Intelligent Systems

In the course Intelligent Systems we have a number of tasks to perform in order to construct an empirical research, which is based on the Google AI Challenge.

Within four weeks we have to go through a process of creating a various amount of bots, which can play the game Planet Wars without human influence.

Four bots are already programmed called the RandomBot, BullyBot, LookAheadBot and AdaptiveBot.

We have to come up with a heuristic and implement it in our bots in order to win from these bots.

We also have to improve our bots, so they can win from any other bot.

In order to program a bot that is up for the task we have to dive in various search methods that we are about to use.

We need a search method that can calculate the best move for us and can predict the move of the opponent, but with a little amount of time and space usage.

During this journey we learned about the different search algorithms and how they work.

We also summarized the advantages and disadvantaged of each algorithm.

This way it was easiest for us to determine the best algorithm for the strategy we would like to apply.

Within the course we start with a template of a already designed bot.

We are able to change the bot according to our heuristic and implement the search methods in these templates.

First it is most important that we get familiar with the game and produce a simple strategy that is the pillar of our final and improved strategy.

So In the first week we have to design a bot that is quite simple and should play a serial game.

This would mean that our bot has to wait for the turn of the opponent before he strikes.

The environment does change every turn, because changes are made.

Planets that where neutral our last turn, would now by occupied by Player Red.

In this weak we produced a bot called GrowthRate.

In week two we had to predict we strategy of the opponent so we could calculate our best move in the beginning and follow that strategy that would lead us to victory.

Of course it is best to calculate a strategy from the beginning of the game, then deciding each turn what to do now without a strategy.

We used the Min-Max method to do this and this bot we called the MinMax.

In week three and four we had to focus on a bot that could understand the strategy of the opponent and react to it.

We wanted to keep close to our roots and therefore we used the strategies in the simple bots and changes them in more intelligent bots that would adapt during the game.

Our problem and research will be discussed more elaborate in the next section of this paper.

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

# Research question

Are adaptive and look-ahead strategies always better than "dumb" strategies which don't look ahead or react to the strategies of the adversary?

# Experimental Setup

## Sort of summary of all bots, but also sort of introduction.

As mentioned in the introduction, we got four already programmed bots to work with; RandomBot, BullyBot, LookAheadBot and AdaptiveBot.

We decided to use the RandomBot and BullyBot in our research. The LookAheadBot and AdaptiveBot however, we changed into our own bots, and we won’t be using these preprogrammed bots in our research.

Now, aside from these preprogrammed bots, we built some ourselves to compete against these preprogrammed bots. We built five additional bots: GrowthRateBot, WeakPlanetBot, MinMaxBot, AdaptiveMinMaxBot, AdaptiveBullyRandomBot.

Next we will explain per bot how exactly it works and why we think that specific strategy will work.

### GrowthRateBot

Our GrowthRateBot is one of the most simple and dumb bots we have built. It doesn’t look forward, and it only depends on one single strategy.

The GrowthRateBot depends entirely on the growth rates of the planets.

Every turn it first finds its own strongest planet; where strength is defined by the number of ships a planet has. It takes this planet as the source planet. From here the ships will be sent to the destination. Than it goes through both the list of enemy planets and neutral planets separately and makes two new lists; one with enemy planets we can conquer and one with neutral planets we can conquer.

Now it can be possible that out of all the planets on the map, there isn’t a single planet we can conquer. So our bot will check both lists to see if they are empty; both lists empty means no conquerable planets on the map.

If both lists are empty, our bot takes all the planets, both enemy and neutral, for the next step, because it still has to choose a planet to attack.

We now have two lists, enemy and neutral, each with either all planets or only conquerable planets. In these lists, our bot now finds the ones with the highest growthrate.

This give us two planets to choose from. Our bot now compares the growth rates of both planets. Because we would rather conquer the enemy planet than the neutral planet, we deduce the growth rate of the neutral planet by one. Than we take the planet with the highest growth rate and chose this one as our destination.

We think this strategy of choosing the planet with the highest growthrate is effective, because if we conquer all planets with the highest growthrate, we get relatively more ships. Having more ships means that you can conquer more planets and this is exactly what you want.

### WeakPlanetBot

Our WeakPlanetBot begins every turn very much the same as our GrowthRateBot; it starts by finding its own strongest planet. Here strength is also defined by the number of ships a planet has. It takes this planet as the source planet.

The WeakPlanetBot’s next step is based on the GrowthRateBot, but with an addition to it.

Before it goes to looking for the highest growthrate, it makes a list of all attacked planets throughout the game until this turn. If this list isn’t empty, it looks for the last attacked neutral planet. We want to attack this planet, because our enemy has wasted his ships trying to conquer this planet, which obviously hasn’t worked; else it would be an enemy planet. The amount of ships on this planet will be relatively low, because the ships the enemy sent to this planet has destroyed an equal amount of the neutral ships.

Of course we don’t want to give the enemy the same opportunity, so the bot won’t take this planet as a destination, if it isn’t conquerable.

The bot now looks to the destination planet, if we have one, it issues the order to attack. If not, it goes through the same steps as the GrowthRateBot to find an destination planet.

So, it finds all conquerable planets, depending on those either takes all planets or not, and then picks the planet with the highest growthrate.

Than we do have a destination planet and our bot will issue the order to attack.

We think this strategy is effective, because you let the enemy waste ships on neutral planets he can’t conquer and then you take them for yourself.

### MinMaxBot

This bot is really an outsider between all our bots. It doesn’t really relate to the GrowthRateBot and WeakPlanetBot. This bot is an adaption on the given LookAheadBot This is also the first “smarter” bot; it looks ahead and simulates the enemy’s moves. The only disadvantage this bot has, is that it always assumes that the enemy is a BullyBot. This because the MinMaxBot isn’t adaptive.

Our MinMaxBot starts with considering each possible pair of own and other planets and taking these as the source and destination.

The bot automatically skips the pair if the source planet has only one ship. But for all other pairs it simulates the next couple of turns. It does this by first simulating its own turn, second, simulating the growth of the planets that would occur in the real game. Then it will simulate the opponents turn, assuming the opponent will do the same as the BullyBot will do. And so on and so on, until the amount of turns to look ahead we gave it is matched.

It calculates the score of the end situation and the source-destination pair with the highest score, it takes as definite source and destination.

Now the calculation of the score is the most influential part, because this is where you decide what you define as a good score.

We thought a long time about what we would define as a good score. Eventually we decided to use the following formula:

In this formula we take the number of our planets and divide that by the number of enemy planets, we also take the number of our ships and divide it by the number of enemy ships. Then we multiply these numbers. This will give us a ratio of the number of planets and ships. This number will be greater if we have more ships or planets.

We think this is the most effective strategy to calculate the score, because it takes into account both the number of planets as the ships. We also think this is most effective, because we have tried lots of different strategies out on paper and this one has proven to be most effective

### AdaptiveMinMaxBot

This is the first smart bot, which does the same as the MinMaxBot does, but without the disadvantage the MinMaxBot had. Namely, this bot does look at the situation of the game and then deduces what kind of strategy his enemy is using.

This bot also is an adaption on the given AdaptiveBot. We just made it a little smarter by giving it the advantage of being able to look ahead.

This bot does the same thing as the MinMaxBot, but where the MinMaxBot just assumes the opponent is a BullyBot, this bot will deduce at the hand of the game situation what kind of strategy the opponent uses. Then it will simulate the opponents turn based on the strategy the AdaptiveMinMaxBot has just deduced the opponent will have.

This give the huge advantage that you can be prepared to what other opponents will do.

Of course it is impossible to implement a solution for every possible strategy the opponent can have. So we kept to the two bots that can be deduced in the preprogrammed AdaptiveBot; BullyBot and RandomBot.

Now the flaw in this AdaptiveMinMaxBot is that it doesn’t really have a good strategy when it has deduced its opponents strategy. If the opponent is a BullyBot it will simply act as a BullyBot in its next turn and the same for the RandomBot.

We think this strategy is more effective than the last three bots we’ve discussed, because it can be prepared for the strategy of the opponent. We also think this won’t be the most effective strategy, because it hasn’t got very good solutions for dealing with a BullyBot or RandomBot.

### AdaptiveBullyRandomBot

Our AdaptiveBullyRandomBot is the smartest of all our bots. This bot is based on the AdaptiveBot, but without the flaw we saw in the AdaptiveMinMaxBot.

So this bot is adaptive, but it doesn’t look ahead. When it determines which planets it will attack, it first goes to determine which strategy the opponent has. Here we also chose to keep the two options of the preprogrammed AdaptiveBot; RandomBot and BullyBot.

We now tried to fix the flaw we saw in the AdaptiveMinMaxBot; the bot doesn’t have good solutions for dealing with the determined bots.

So we thought what for both the opponent strategies would be good solution. With the RandomBot it really was easy, because RandomBot does what its name predicts. It takes random planets to attack. So here we implemented as solution our GrowthRateBot turn.

The BullyBot turn was a little more difficult. But when you start thinking about it, what would you do when you had to defeat a bully? You would try to make him less powerful. Because a powerless bully is no bully.

This is exactly how we implemented the solution. If the opponent is deduced to be a BullyBot, our bot would take its own strongest planet and attack the opponents strongest planet. This to try and deduce the number of ships the opponent has.

We think this is a good solution, because the less ships an opponent has, the less damage it can do to your smallest planets.

# Results

In this part we will discuss the results of our experiment. We decided to test every bot against every other bot and do this for three different maps; one with eight planets, one with twelve planets and one with sixteen planets. We let each ‘battle’ between two bots in a map be fought five times to get a steady result. In figure 1, 2 and 3 you will find the results of the experiments. They exist of a table with the number of losses and a table with the amount of turns a battle cost on average. The turns are counted as the amount of turns displayed beneath the Planet Wars map when you run a game.

//FIGURE 1

In this figure we can find the results of the battle in a map with eight planets. The numbers represent the amount of games won by the bot in that row against the bot in that column. So if we want to see how many times the MinMaxBot won from the BulyBot we look on the MinMaxBot’s row (the fifth) and in the BullyBot column (the second). This way we will find out that the MinMaxBot won 5 times from the BullyBot.

As we can see there are a lot of draws, represented by ‘—‘. If we look at the amount of turns these games took we will find that the games probably ended in a draw because there was an infinite loop going on. The amount of games is also in green, which means that the amount of turns was constant for every one of the 5 games played. The only results that are not green are the games played by the RandomBot, which is perfectly explainable because the RandomBot is doing random turns, whereas the other bots all have strategies and will do the same turn in the same occasions.

Another interesting result is that the results of the GrowthRateBot and the WeakPlanetBot are exactly the same (except when played against the RandomBot). This is also perfectly explainable because the GrowthRateBot and the WeakestPlanetBot are in principal the same bot, except for the fact that the WeakestPlanetBot will first search for a neutral planet that has just been attacked. In most cases there are no neutral planets that have been attacked because the other bots are mostly looking for enemy planets to attack.

The last interesting result is that the RandomBot has won no game at all and the MinMaxBot, AdaptiveMinMaxBot and AdaptiveBullyRandomBot have won all their games.

//FIGURE 2

Figure 2 contains the results of the 12-planet maps results. The biggest difference with the 8-planet map results is that there are no draws. Other than that there are no major differences. The amount of turns is slightly higher overall but that is explained by the higher amount of planets. It is likely that a player needs more turns because there are more planets to chose from.

//FIGURE 3

Figure 3 shows the results of the 16-planet maps results. The results are almost the same as for the 12-planet maps, but one thing that is noticeable is that the amount of turns is slightly lower this time. Especially with the MinMaxBot and the AdaptiveMinMaxBot.

# Findings

To get an answer to our research question we will have to look at our results and find the relevant and interesting parts of it. Therefore we collected the total amounts of wins and losses of all maps and divided them into four different categories. These results can be found in figure 4.

//FIGURE 4

As we can see, the amount of wins of the adaptive non-look-ahead bot is significantly higher than the amount of wins of the other bots. The non-adaptive non-look-ahead bot has the lowest amount of wins. The look-ahead bots both have almost the same amount of wins.

These results are also in line with the amount of losses. The adaptive non-look-ahead bot has only lost one game, whereas the non-adaptive non-look-ahead bot has lost 50 games. Between the look-ahead bots there can be seen a bigger difference in the losses than in the wins. The adaptive bot has more than twice more losses than the non-adaptive bot.

From these results we can conclude that the Adaptive non-look-ahead bot is the best bot we built. We can not say that adaptive bots or look-ahead bots are per definition better than non-adaptive and non-look-ahead bots. We can only say that the combination of non-adaptive and look-ahead and the adaptive and non-look-ahead work the best.

# Conclusions

1. <http://www.encyclo.nl/begrip/kunstmatige%20intelligentie> [↑](#footnote-ref-1)
2. <http://nl.wikipedia.org/wiki/Kunstmatige_intelligentie> [↑](#footnote-ref-2)
3. <http://planetwars.aichallenge.org/> [↑](#footnote-ref-3)
4. <http://en.wikipedia.org/wiki/AI_Challenge> [↑](#footnote-ref-4)
5. <http://www.franz.com/services/conferences_seminars/webinar_1-20-11.lhtml> [↑](#footnote-ref-5)