Annales UMCS Informatica AI XII, 1 (2012) 39–45
DOI: 10.2478/v10065-012-0009-z

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Automatic reasoning in the Planet Wars game

Bartosz Ziółko[1*], Maciej Kruk[2]

[1] *Department of Electronics, AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland.*
[2] *Department of Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland*

Abstract – An artificial intelligence algorithm for a computer game competition organised by Google and University of Waterloo is presented. It competes with others by reasoning, evaluating of a situation and taking decisions in a war simulation.

1 Introduction

Computer games are rapidly growing computer science industry all over the world including Poland. It is also a popular field to test and develop several types of AI (artificial intelligence) algorithms [**1, 2, 3, 4, 5**].

The objective of the Google AI Challenge [**6**], a programming contest organised by the University of Waterloo Computer Science Club, was to create an autonomous computer program that plays the game of Planet Wars as intelligently as possible. Planet Wars is a strategic game set in the outer space based on Galcon. A game takes place on a map (let (i, j) be discrete coordinates) containing several planets, each of them with a specific number of space ships on it, belonging to one of the two players or neutral. In each turn, the player may choose to send fleets of ships from any planet he owns to any other planet on the map. The game includes a certain maximum number of turns, which is 200. The player with the more ships at the end of the game wins.

The Galcon game consists of a map and ships. Planet positions are specified relative to a common origin in the Euclidean space. The coordinates (i, j) are given as the floating point numbers. However, to simplify mathematical description, let us see the map as a matrix $M[i, j]$. Planets never move and are never added or removed as the

*bziolko@agh.edu.pl

game progresses. Planets are not allowed to occupy exactly the same position on the map. The integer values m_{ij} of$M$ describe production of (i, j) position of the map. The values $m_{ij} = 0$ represent empty space. Planets are locations fulfilling $m_{ij} > 0$ and they can either belong to one of the players or are neutral. Matrix M does not change in a particular game and even though it is not mentioned in the rules, it is symetrical in all games to allow equal chances of winning for both sides.

In each turn, the player may choose to send fleets of ships from any planet he owns to another planet on the map. He may send as many fleets as he wishes in a single turn as long as he has enough ships to supply them. After sending fleets, each planet owned by a player (not owned by neutral) will increase the forces there according to the planets growth rate $m_{i,j}$. Different planets have different growth rates. The fleets will then take a number of turns to reach their destination planets, where they will then fight any opposing forces and, if they win, take ownership of the planet m_{ij}. Fleets cannot be redirected during travel. Players may continue to send more fleets in later turns even while older fleets are in transit.

The growth rate of the planet is the number of ships added to the planet after each turn. If the planet is currently owned by neutral, the growth rate is not applied. Players can only get new ships through growth. The growth rate of a planet will never change. It is given as an integer value. Let us denote $A[i, j]$ and $B[i, j]$ as matrices describing the positions and numbers of ships for both players. A and B change their values during all turns. The first type of changes is due to production of new ships on planets

$$\begin{aligned} \forall_{i,j}\ a_{ij} &:= a_{ij} + m_{ij} \text{ if } m_{ij} \text{ is controlled by player } A \\ \forall_{i,j}\ b_{ij} &:= b_{ij} + m_{ij} \text{ if } m_{ij} \text{ is controlled by player } B\ . \end{aligned} \tag{1}$$

The second type are movements. Players can order any movements of their ships a_{ij} and b_{ij} to any positions (i, j) such that $m_{ij} > 0$ (namely to other plantes). A transfer cannot be cancelled once started, however, it will last several turns and ships can be located on any (i, j) position while moving. The total trip length is given as an integer, representing the total number of turns required to make the trip from the source to the destination. The remaining turns are also an integer, representing the number of turns left from the current turn to arrive at the destination. Trip lengths are determined at the time of departure by taking the Euclidean distance to the destination (i, j) from the source (k, n) and rounding up, w (noted later on by $\lceil\rceil$). Finally, if ships of opposite players $a_{ij} > 0$ and $b_{ij} > 0$ meet on the planet $m_{ij} > 0$, they fight. There is no random aspect in such fights. If $a_{ij} > b_{ij}$, then next turn $a_{ij} := a_{ij} - b_{ij}$, $b_{ij} := 0$ and A takes control of production m_{ij}. If $b_{ij} > a_{ij}$ the other way around.

2 Basic algorithm

Our solution divided possible changes in a game state (moves) into several groups, based on the history of ownership and location $m_{ij} > 0$ of the targeted planet:

(1) Defence

(2) Attack
(3) Expansion
(4) Supplies

Before these moves, planets, states in the future are evaluated in each turn.

Every turn, AI makes a decision, which of these moves is locally the highest gain value and performs them. To make the decision process more accurate, it needs a whole range of data concerning the planets and fleets:

(1) General planet specification (location, size) $M[i, j]$,
(2) Planet status in each point of time i.e.: ownership (A, B or neutral), number of ships including incoming fleets ($A[i, j]$ and $B[i, j]$),
(3) Possible ways of attack (distances between $m_{ij} > 0$),
(4) Planet location in context of other planets of its owner.

At the beginning of each turn, the program calculated all the necessary information. Next it entered a sequence of functions responsible for fleets discharge (defence, attack, expansion, supplies) in which it assessed each planet's value and based on it and surrounding planets state made local decisions of ships transfers.

3 Reasoning and decision taking

The function of predicting the future calculates planet states in the future based on planet ownership, growth rate and flying fleets. Additionally, the function computes how many ships can enemy deliver to each planet in each step (used for attack prevention).

Defence of planets under attack and ships surplus calculation is taken by the defence function. It ensures that after

$$max\_distance = \max_{m_{ij}>0, m_{kn}>0} \left\lceil \sqrt{(i-k)^2 + (j-n)^2} \right\rceil \tag{2}$$

turns, as many as possible of its planets will remain under control.

For each of its planets, the algorithm calculates the number of ships needed to keep them from being overtaken by the enemy. Later it sorts them according to their production rate m_{ij}, distances to other planets $\left\{ \left\lceil \sqrt{(i-k)^2 + (j-n)^2} \right\rceil : m_{kn} > 0 \right\}$ and distances to the enemy $\left\{ \left\lceil \sqrt{(i-v)^2 + (j-w)^2} \right\rceil : m_{vw} > 0 \right\}$ and try to defend as many as possible. If it is impossible to keep the planet during all time, the algorithm tries to retake it, not long after it is overtaken. If the planet still belongs to the AI during the next (2) turns, it calculates how many ships it can send off from this planet not losing them.

The attack stage consists of two activities. The first one is sniping (see Fig. 1). The algorithm tries to attack neutral planets that will be overtaken in the future soon after the attack. In this way, the enemy loses his forces to fight the neutral planets and our algorithm can overtake a planet with fewer ships.

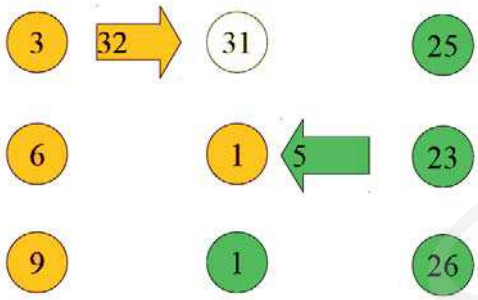

Fig. 1. Sniping is a strategy of waiting for conquering a planet by an enemy losing ships during on its defence. Left planet belongs to b and has 35 ships, right to b with 25 ships, and the central one is neutral. First, the player b attacks the central planet using 32 ships. 31 of them are lost in fight with the neutral forces. Afterwards the player a sends a fleet to fight the only survived ship of b plus what he will manage to produce on the central planet, which is 3 in this example. In the last stage the central planet is in control of a and what is very important, the player a has more ships, namely 26, on its starting planet and player b, 9. The player b has no chance to win, even though, he starts with more ships than the player a

The second phase is a regular attack. The algorithm browses through all enemy planets and attacks those it is sure that it can overtake even if the enemy sends all the ships to defend it (even for just one turn)

$$m_{ij}^{fitness} := c - b_{ij}^{t} + 1000 m_{ij} \tag{3}$$

where b_{ij}^{t} is the prediction of the value of enemy ships in t turns in the future on $m_{ij} > 0$ and $c = 2\ 000\ 000$ is a constant, large value. Value of t is the distance $\left\lceil \sqrt{(i-k)^2 + (j-n)^2} \right\rceil$, where m_{kn} is a planet from which we plan to attack. The algorithm avoids attacking the planets further than $0.5 max\_distance$, unless there are no enemy planets closer.

The expansion phase consists of neutral planets evaluation (based on their growth rate, number of ships and distance to both players).

$$m_{ij}^{fitness} := m_{ij} \left(\left\lceil \sqrt{(i-k)^2 + (j-n)^2} \right\rceil - \left\lceil \sqrt{(i-v)^2 + (j-w)^2} \right\rceil + 1 \right) - s_{ij} \tag{4}$$

where m_{kn} belongs to the enemy and m_{vw} to our AI and s_{ij} is the number of neutral ships on m_{ij}. Positive value of (4) means it will be advantageous to take over m_{ij} even if the opponent will take it over afterwards. If (4) is negative, other arguments are taken into account.

AI expands more aggressively, if the own accumulated growth rate is lower than that of the enemy

$$m_{ij}^{fitness} := m_{ij}^{fitness} + \begin{cases} 50 m_{ij} & \text{if winning} \\ 20 m_{ij} & \text{otherwise} \end{cases} \tag{5}$$

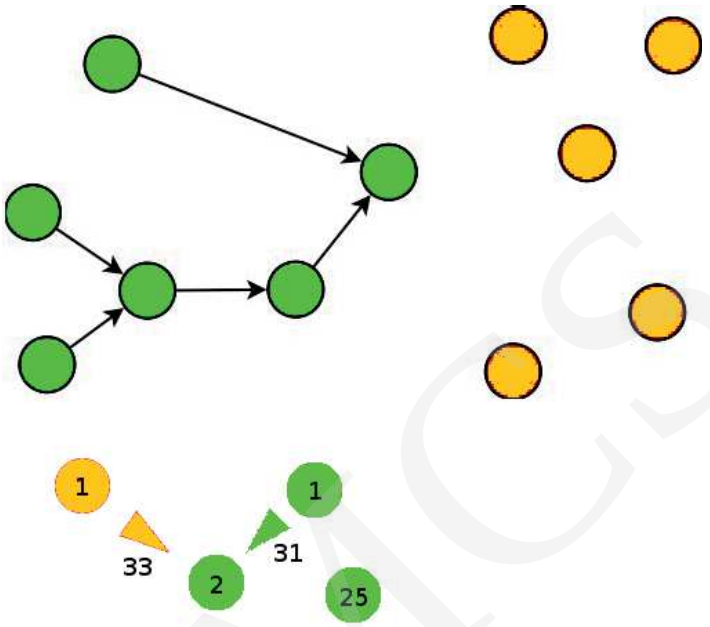

Fig. 2. Safe planets should supply the frontal one with ships. The reason for it is that the planets far away from the oponent are unlikely to be attacked. Even if they are attacked they could be reinforced, between discharging ships by the enemy and arriving to the target. In contrast, typically there are front line planets which cannot be reinforced already during the attack because the distance from the enemy is smaller then from reinforcing planets. What is more supplying scheme is also useful to gather ships for a full attack with as short warning as possible

The map is always symmetric. AI prefers the planets whose mirrors are overtaken by the enemy (its $m_{ij}^{fitness}$ is increased by 1 000 000). The algorithm always attacks the planets it is sure to profit from. If it has enough ships, it will also try to overtake other planets, based on their previously calculated value $m_{ij}^{fitness}$.

The front consists mainly of the own planets (sometimes also neutral ones that are attacked) that can not be defended by sending ships on time from other planets in case of enemy attack. At the same time, they are capable of defending any non-frontal planet of our AI. Storing all own ship reserves on the frontal planets improves our defence and attack potentials.

Another part of the algorithm is reasoning about supplies (see Fig. 2), which is a movement of surplus ships from the non-frontal planets to the front ones. The number of ships sent to each planet is proportional to the sum of reciprocals of distances to the enemy planets multiplied by the numer of ships they store. This way it tries to take into account enemy ships movements.

4 Conclusions

The described algorithm is ranked as 105 for over 4600 competitors and the sixth in Poland. The most difficult task in such AI is in providing a balance between defence, expension on neutral planets and attacking the enemy. Each game consists of too few turns to adapt behaviour to a particular enemy, especially that calculations have to be made in up to 2 seconds per turn. Hard coded rules are often too defensive for some opponents and maps while too aggresive for others.

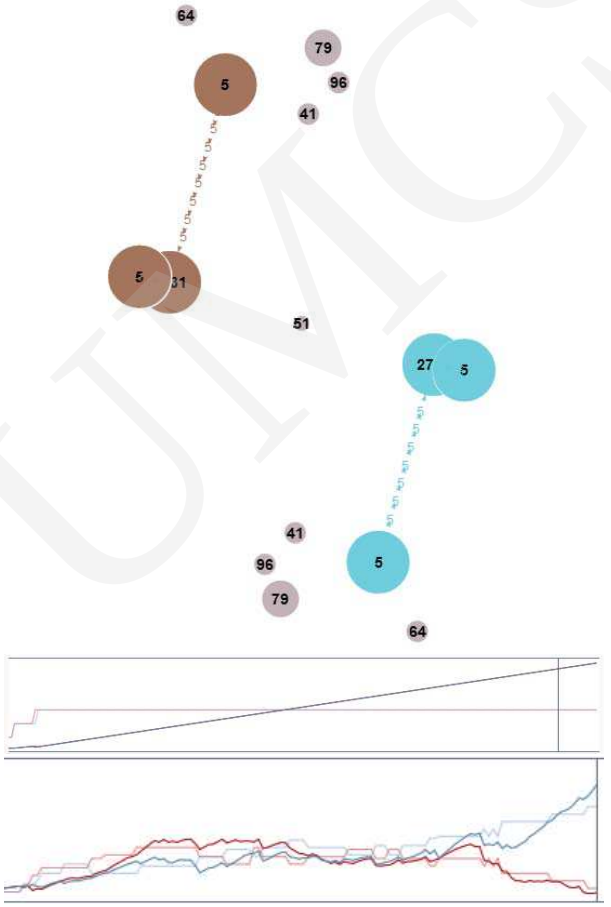

Fig. 3. Example of a game with too little expansion (top) and linear, correlated functions of growing of both opponents (middle). The functions summarising another (more typical) game are presented for comparision(bottom). Both AIs will keep collecting ships in the case of an attack which is never going to happen. An AI which would decide to take a risk and attack a neutral planet to enlarge its production potential is likely to succed in such a scenario.

Undoubtedly, the hardest part to balance is the expansion phase. Too little or too late attacks on neutral planets usually mean AI is left behind in terms of ships production (see Fig. 3). Too many or badly chosen expansions can make one vulnerable to enemy attacks, or give away the attacked planet to an enemy.

Instead of making local decisions to attack/expand/defend, it would be probably better to gather all data about every planet, rate them taking into account their interrelations and then decide on the fleets destination. Also instead of finding the source-destination-ships combination, it would be better to plan the planet population and then send fleets from the closest planet to achieve it. That should help fight off the situations when fleets are being sent to defend distant planets.

In the case of attacking a planet from a few different sources, instead of attacking it simultaneously, it would be better to take into account their distance to the enemy and dispatch fleets at the last possible moment, so all of them reach their destination at the same moment.

Some enemy movements are easy to predict (e.g. planet sniping, repeated supplies, constant attacks) and hence the should be taken into account when making decisions. Other types of movement can be taken into account by browsing the game tree. It is too big to analyse thoroughly (the competition gives a calculation time limit for each turn) but at least the analysis of the potential attack routes should be possible. The main problem is in including these predictions into the future states analysis.

Acknowledgements

Some figures were made by Jakub Gawlik thanks to the organiser and the sponsor of the competition.

References

[1] Baba N., Jain L. C., Handa H., Advanced Intelligent Paradigms in Computer Games, Springer (2007).
[2] Barber H., Kudenko D., Generation of adaptive dilemma-based interactive narratives, IEEE Transactions on Computational Intelligence in Games 1(4) (2009): 309.
[3] Rabin S. C., AI game programming wisdom, Charles River Media, Inc. (2002).
[4] Funge J., Artificial Intelligence for Computer Games: An Introduction, A K Peters, Wellesley, MA. (2004).
[5] Millington I., Funge J., Artificial intelligence for games, Elsevier (2009).
[6] AI Challenge Ants: http://ai-contest.com/ (12.10.2012).