Planet Wars

Artificial Intelligence Lab - Project 1

Anand Dhoot 130070009

Anchit Gupta 13D100032

Charmi Dedhia 130070007

October 12, 2015

1 Introduction

Planet Wars is a game based on Galcon. A game of Planet Wars takes place on a map which contains several planets, each of which has some number of ships on it. The goal of the game is to make a bot which returns a set of moves in each turn, given the Game State. The final goal is to either kill all enemy ships or to finish with more ships than the enemy.

2 Explanation of the Algorithm

2.1 First Turn Knapsack

The First Turn is the single most important turn which pretty much sets the tone for the remaining war. Hence, playing the first turn in the best possible manner is essential. as a Knapsack Problem.

Our algorithm consists of first finding the list of neutral planets which are "un-snipable". We also calculate the number of spare ships we have on our home planet considering the worst case scenario when the enemy does a RageBot style all-out attack against us.

The cost of each planet is taken as $NumShips + 1$. Whereas the value is taken as $(100 - dist)growthRate$.

Often, a common choice is to maximize the growth rate of planets. However, we chose the above function instead, which seemed to give better results. We believe this is because we want the fleets to target large planets which are close-by so that the fleets reach the destination quickly, start producing more ships and subsequent algorithm can kick in.

Now we need to maximize the total value of chosen planets subject to constraint that the $totalWeight < ReserveShips$. This is a instance of the well-known Knapsack Problem [1].

2.2 Maintaining a Timeline

We make a timeline for each planet that keeps track of the estimated number of ships and the owner of the planet from the current time till the horizon. This is done taking into account the fleets currently in flight and the growth-rate of the planet, wherever required. All cases pertaining to owner changes are also considered.

A new timeline is made in every turn to account for the new fleets that are sent. We realized that maintaining the same timeline would be much more challenging than creating one from scratch in each turn. This idea was inspired from [2].

2.3 Planet Reserve Calculation

For planets that remain in our control at the horizon, the number of reserve ships get calculated. We define reserves to be the number of ships which can be sent away in the current turn such that even after sending them, the planet will not be lost in the future. This computation is done using the timeline and is used in subsequent steps in the algorithm.

2.4 Defending

This step involves finding vulnerable planets and issuing orders to ensure that they get defended. Vulnerable planets are the ones which were in our possession at a certain point of time and are owned by the enemy at the horizon.

For every such planet, we create a "Defense Task" corresponding to each incoming fleet after the planet gets captured by the enemy. The intuition behind this is that if we are able to satisfy the "Defense Task", we can ensure that the planet remains in our control.

We then proceed to try and satisfy the above computed "Defense Tasks" in a Greedy Manner. For each Task we try and formulate a set of Fleets from the reserves calculated earlier, such that they should be able to reach before the fleet we are defending against.

The reserve ships get updated due to the executed Defense Tasks and are carried forward to subsequent steps.

2.5 Abandoning planets that cannot be defended

In the above step, we also identify our planets that cannot be defended for various reasons. The ships present on the planet would get wasted anyway as the planet goes out of our possession. However, if used for an attack, we might end up getting some additional planet and hence its growth rate for subsequent turns. Hence, we add the number of ships contained by a such a planet to the reserves for attack.

2.6 Attack

Each Planet with a non-zero reserve force tries to formulate a attack against each Planet which is not ours at the horizon. Every such attack is assigned a score and the planet executes the attack with the best score.

The number of ships required to take over a planet takes into account the incoming enemy fleets etc. using the timeline. The potential defense forces the enemy might muster in response to our attack is calculated and an attack is executed only if we have more ships than this requirement.

The scoring function used is

$$Score(p) = \begin{cases} \dfrac{growthRate(100 - dist) - requiredShips}{requiredShips} & p \in Neutrals \\ \dfrac{2 * growthRate(100 - dist) - requiredShips}{requiredShips} & p \in EnemyPlanets \end{cases}$$

. This ensures that larger planets will be given more priority for attack. Also, dividing by requiredShips favors attacks which need lesser number of ships and thus leave out more ships for future defense and attack tasks are carried out first.

As a final check we see if the best score of a attack is above a threshold and only execute the attack if it is so, this prevents a few bad attacks from happening

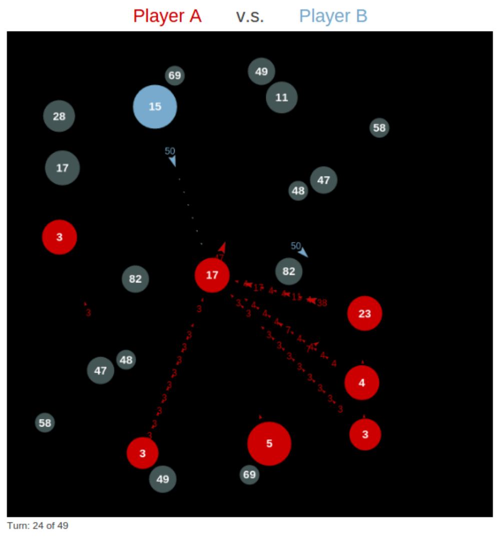

Figure 1: Planets sending ships to the frontlines

2.7 Move to the frontlines

For every planet, the ships in the reserve that remain unutilised in the previous step are sent to the ”frontlines” - our planet closest to the closest enemy planet.

If the ships are moved to the frontline using an alternative path where at each step, they are moved to the closest friendly planet in the direction of the frontline, the ships would be airborne for several small periods of time. This leaves us with more options in the future.

Sending ships to the frontlines ensures that our ships remain closest to where the action is happening as well as being available to defend any incoming fleet on a planet which is not on the frontlines.

3 Possible Improvements

3.1 Move splitter

This involves sending the ships in phases. Consider a case where you want to send a fleet from planet A to planet B and planet C lies exactly in between the two. When using move splitter, the order to send fleet from planet A to planet B is broken down into two orders - to move from A to C and a "Future Order" to move the fleet from planet C to planet B, once it has reached planet C.

We analysed our performance on implementation of this algorithm and realized that a complete implementation of this would require a lot of bookkeeping.

3.2 Multi-Planet Attack

We also made an implementation where multiple planets would choose a single target and attack in unison to capture it. This would in theory open up a lot more potential attack then the above implementation. In Practice this seemed to make our bot a bit too aggressive resulting in it often using up too many ships leaving itself weakly defended in subsequent turns.

A potential fix to this would be to do a more intelligent calculation of Planet reserves which also takes into consideration the the Attack/Defense potential of a planet[2] .

These two features together we think would strike the right balance between attack, defense and make the bot much stronger.

3.3 Ship streaming

If we are ahead in bot shipcount, production and not able to generate a good attack move. It might still be in out favour to stream some ships to a enemy planet hence reducing both our ships by the same amount and potentially reducing our opponents choices in the future.

4 Cases when the Algorithm Excels

This was analyzed by varying maps and opponent bots, one at a time, keeping the other one constant.

4.1 Varying maps

When running on various bots on the same map, we observed that our bot performs better when one of the following two things happen -

- There is a even distribution of planets across the map. This means that the planets are neither too far apart, nor too close forming large clusters
- When the enemy's home planet is far away from our home planet.

Our bot is more defensive and less aggressive. We do well in the first case because when the planets are evenly distributed, the bot striking a balance between attack and defense gets the upper hand. Clustered maps favour defensive strategies and spread out maps favour attacking ones.

We believe that we win more often when the enemy planet is far away because in such cases it takes much more time for a opponent's move to start directly affecting us.

Moreover, both the above conditions make our timeline based calculations closer to the real deal. Small distance between planets results in added unpredictability.

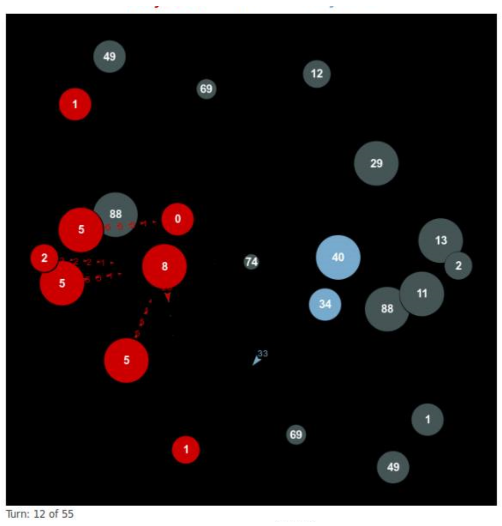

Figure 2: Fast Expand

4.2 Varying enemy bots

Our bot performs better against aggressive bots which attack without taking into consideration our defense (RageBot, for example). The fact that we have a solid defense strategy results in such bots wasting their ships and leaving themselves vulnerable to our attacks. Such matches usually end very quickly.

We perform well also against bots that are slow to expand. As we have a very good expansion strategy if the opponent is not able to get a comparable growth rate to us in the first few turns, then our bot is very good in winning quickly.

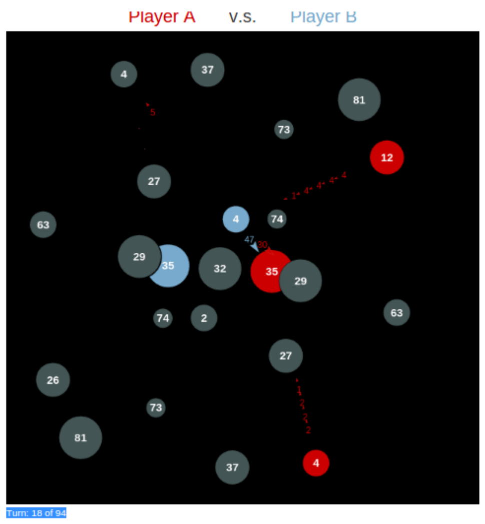

Figure 3: Initial State

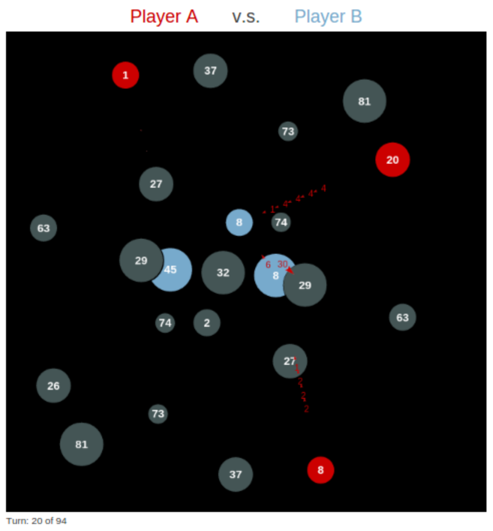

Figure 4: Final State

Close Starting State against aggressive bot

4.3 General scenarios

Our attack algorithm inherently scores sniping opportunities highly. This means that bots which do snipable attacks to capture neutrals become vulnerable.

5 Cases when the Algorithm Fails

This analysis was also done by varying the maps and opponent bots, one at a time.

5.1 Varying maps

Our algorithm fails when the enemy's home planet is close-by on the map and it starts performing RageBot-type attacks/snipes.

The reason for this is that we expand in our first step, capturing as many planets as possible. However, in the process of doing this, most of our ships get destroyed battling the neutral ships. Further, proximity of the enemy means that its attacks reach us faster. In cases where the enemy directly attacks us, instead of the neutrals, we are left with no forces to defend. Also, our timeline does not remain as accurate in predicting the future.

5.2 Varying enemy bots

Bots which take into account opponent moves also while playing our often able to do much better against us. Bots which redistribute their ships anticipating our best moves to cover their vulnerable planets[2] rather than just simply moving to frontlines are much more solid than ours.

5.3 General scenarios

If the planets expanded to in the first stage are far away from our home planet, the fleets remain in air for a long time. This means that defensive forces take time to arrive and the defense does not remain very effective. In such a scenario, the enemy catches us in a vulnerable position.

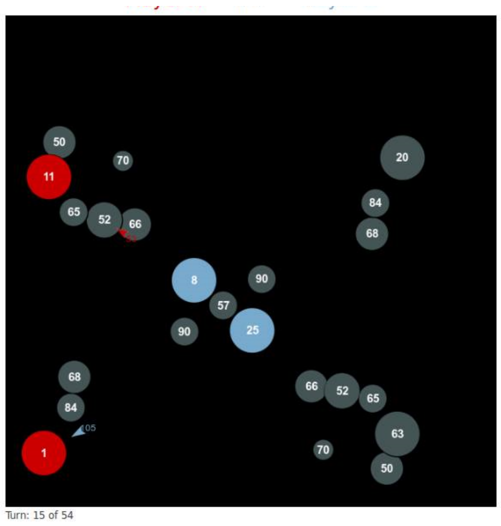

Figure 5: Planets far apart and cannot support each other in defense

6 Source(s) of Inspiration

The idea of using Knapsack was from various blogs and is specifically also present in [2]. The Knapsack code we came up with was a modification of the generic code [1].
The idea of maintaining a timeline was also taken from [2].
The scoring function we used for both Knapsack and Attack was original.
Moving to frontlines and Defending before Attack was a idea inspired from [3].
This Blog served as guide for us especially in the early stages while designing the bot as it was written in a way that was simple and easy to grasp.

References

[1] Generic Knapsack Code

[2] Second Ranked Bot's Post-Mortem

[3] Blog of a Top-50 Bot(Knights)

[4] Google AI Challenge Main Website