Planet Wars

Samarth Mishra  
Katuri Sai Kiran  
Ravichandra Addanki

130260018  
130050051  
130050061

October 12, 2015

Pro-loggers
Artificial Intelligence
Dept. of Computer Science
IIT BOMBAY
Abstract

The algorithm and strategies used by our bot include many different sub-strategies like future state simulation, defense strategies, attack strategies, front lining etc.. We explain these parts in each section below. Then we move onto the heuristic estimation techniques used, cases where our bot excels and also the cases in which it fails and conclude by giving the possible improvements and the scope of future work.

1 Future State Simulation

1.1 Motivation

For calculating either attack or capture or defence, we need to have some idea about how the future of every planet is going to be. For instance, if we want to capture a neutral planet with 1 ship, then sending a fleet of 2 ships may not help if there is an enemy fleet of 10 ships directed to the same planet currently.

Another more complex example is if there is a neutral planet N and another neutral planet NP which is very close by N. If in the near future, enemy is going to capture NP, then it is not a good idea to capture N. If we still want to capture N, then we would like to send sufficient number of ships such that even if enemy tries to attack N from NP after capturing it, we could survive.

Expressing all these conditions would be very complex, unless we simulate the future for all the planets for the foreseeable future.

1.2 Algorithm

Implementation/Algorithm of Future State Simulation

1. For every turn till maxTurns, defined as the max distance between any two planets called as horizon do :

2. Set the forces present on the planet to 0 for every player, except the owner, who has planet.NumShips() as forces

3. For every fleet, check if the fleet arrives at the current turn. If so, add the fleet to the total size of the fleet owner’s forces
4. Apply the battle resolution logic (largest force minus second largest force is remaining number of ships on the planet. If remaining ships > 0, the new owner is the owner of the largest force)

Additionally it is possible that we may want to re simulate the future after issuing few orders. Then it is necessary to re-simulate the future based on the orders issued. We provide this flexibility by allowing the user to provide vector additionalFleets which will be added to the current set of our fleets.

However we have no idea about what the enemy’s moves are going to be in the future. So we simulate the future only based on the moves made by him in the past. Additionally, we also provide a method to re simulate the future of a single planet, and not the whole planet set. We used this method in calculating the defense moves.

1.3 Code

Please find the code of above simulation in the file FutureState.cc.

1.4 Credits

This is not entirely our own idea but we were introduced to this from several posts in the ai planet wars forum. We also learnt this from other teams who used a similar idea with names such as TimeLine. Two posts which presented this idea are Mistmanov[2], Zilog[4] and Khramtsov[1]. Our algorithm is quite similar to the one described in Mistmanov[2], Zilog[4]. However we have provided the additional facilities of having additionalFleets and also re simulating the future of just one planet.

2 First Turn Computation

Some of the static computation that does not change during the gameplay can be done in the first turn. In our implementation, we’ve calculated the maximum distance between a pair of any two planets in this first turn, this would later be used as the horizon till which we compute the future state of some planet in future state computation. Similarly we’ve also calculated maxGrowthRate, minGrowthRate, minDistance.
3 Defense Moves

3.1 Motivation

Defense is obviously one of the most important part of this game. This is because losing one of our planets to an enemy will not only decrease our growth rate, but also adds up to the growth rate of our enemy for the future and growth rate is the most important factor that will eventually decide the winner of the game. This is why we have treated defence as our top priority and used the ships to issue defence orders first before proceeding to expansion. This obsession towards defence benefitted us hugely but at the same time made our bot weak in certain maps. More about this at the end.

3.2 Algorithm

Populating Imaginary Fleets for Defense

For every planet, we define a number called spare, which gives the number of ships it can spare and still not be captured by the enemy in the future.

1. For every planet that is ours do :

2. Simulate its future including all the “imaginary Fleets” (defined below) sent.

3. If at all the turns in the simulated future, we still have control over the planet, then we don’t need to defend it in the current turn. Set the spare of this planet to the minimum number of ships on this planet (over the computed future) and move on to the next planet.

4. If at any turn in the simulated future we lose it to enemy, then send an imaginary Fleet with appropriate number of ships(considering growth rate of planet) to take over control at least before the enemy captures it in the simulated future.

5. Give a heuristic to the above sent imaginary Fleet, depending upon the emergency of saving the above planet. Add it to set of all imaginary Fleets.

6. Go back to step 1.
The above described algorithm, checks if in the future any of our planets is in distress, and if so it adds to the imaginary fleets a fleet to be sent by us to rescue it and then re-simulates the future assuming we are able to send this fleet in real.

If for any planet under attack if we are able to send all the above computed imaginaryFleets to it in the future, then we will be able to defend it. If not, we give up on defending this planet. To do the above task we first sort all the above computed imaginaryFleets based on a heuristic and for each one we try to see if it can be sent in reality.

Issuing orders for Defense

sort imaginaryFleets on the heuristic.
For every imaginary Fleet F to be sent :

1. If we have given up on the planet that F is trying to defend then we ignore F and continue.

2. Else, Go through our planets which have positive spare and see if the planet can contribute a fleet as a part of F(can send at a turn at least before F expires)

3. If we have found sufficient number of our planets which can send fleets that add up to produce the imaginary Fleet F, we issue orders from all of the above computed planets. The imaginary Fleet F is now real

4. Else, we cannot make F real. So, we give up on defending whatever planet that F is supposed to defend

The above defined algorithm is a naive implementation of the more complex algorithm described in Defence, Mistmanov[3]. The idea to sort all the defenceMoves on a heuristic (and not issuing them greedily, which we did in our initial version) was taken from Defensive Strategy, Mistmanov[3]. The idea to “give up” on a planet was our own idea and not found above.

3.3 Strengths

- Our defence strategy, uses ships from multiple planets of ours for def-
ence. This is definitely desirable
• The defence takes care of the cases when one planet is attacked from multiple enemy planets with multiple fleets from each very neatly.

• The defence strategy tries to defend the planet till it possibly can and gives up only after then. Even though we ultimately lose control over the planet at the end in such cases, holding it for few turns can increase the number of ships due to growth on that planet.

3.4 Weaknesses

• In the above defence strategy, we don’t try to get back one of our planets that has been captured by the enemy. We give up on our planet once we see that we cannot stop it from being captured by the enemy. So if the enemy captures a planet and maintains it even for one turn, we stop defending that planet. It might be possible to re-capture one of our captured planets but the defence strategy doesn’t do that. (This might be sometimes be done by the offence moves below).

• An example which explains the above shortcoming is as follows: A planet X (growth rate: 1 per turn) of ours having 10 ships at turn 5 would be captured at the turn 5 by an enemy fleet of size 6. Planet Y of ours is at a distance of 6 turns from X, and has 100 ships to spare. We can issue an order to send a fleet of 3 ships from Y to X and get back the captured planets by turn 6. But we don’t do this. We simply give up on this planet in our defence strategy which is not the optimal thing to do.

• We couldn’t see this defect visually in the games that the bot played because this shortcoming is partly taken care of by attack moves (described below). For instance, in the above example, at the turn number 5, we see that the planet X is a very easy target and we issue an order to capture it.

4 Attack Moves

4.1 Motivation

For the attack strategy, we implemented the offensive algorithm described in Mistmanov[1] without accounting for any profits. A similar algorithm has appeared in various places, for example in Zilog[5].
4.2 Algorithm

Algorithm for Attacks

For each of my planets P, and neutral/enemy planets OP do the following:

1. look up the future state of OP at pw.Distance(P, OP) = T turns in the future. Number of ships to capture the planet by turn T would be 1 + number of ships on that planet at turn T

2. We also look at the planets EP which are closer to OP than distance T. If the enemy captures one such EP before turn T, then it might also be possible that the enemy sends a fleet to capture OP, before turn T, thus making our attack move weaker

3. To counteract the above shortcoming, we add to the number of ships to send to capture OP, the number of enemy ships on EP which can potentially be sent to attack OP by turn T

4. If we can send so many ships without endangering P itself, we calculate a heuristic score that indicates the advantage that we might get from such an attack move

5. We take the attack move with max score for all OP, and carry it out if it exists. We re simulate the future based on the attack order issued

4.3 Shortcomings

- The above strategy cannot attack more than one planet from a given planet in a single turn. This is a drawback for the attack from the front-line planets which usually have a lot of ships (due to front-lining) and which are very close to enemy planets

- For instance, consider one of our planets X which has 100 ships and its neighbours are two enemy ships Y, Z each having 10 ships at a distance of 10 turns each. While we can launch an attack in the same turn to capture both, using the above strategy, we have to wait for the next turn to launch the second attack.
• However this defect’s impact is not seen much in the practice, because the second possible attack from a planet can be made in the immediate next turn during which the game state would have been approximately the same.

4.4 Possible Improvements

• In the above algorithm, we don’t consider the profit gained from the attack to be made as suggested in Mistmanov[1].

• An improved version of the above attack strategy would be to include a cost estimation, which gives us the max number of ships that can be gained due to growth rate before losing control of the captured planet minus number of ships to be sacrificed to take over the planet.

5 Front line reinforcement

5.1 Motivation

It is always a good idea to have more number of ships on our planets which are closer to the enemy than on the planets which are behind. This makes us less susceptible to attacks and also makes it easier to conquer an enemy planets if and when it becomes weak.

5.2 Supply route

The generation(growth) of ships happens on all the planets even the ones that are not on the frontline. So we would like to “pass” the ships that are being generated over time at the planets that are far from the enemy to our planets that are closer to the enemy. The route we take to pump the planets forward is called supply route. The above term was first described in temp_dummy[5].

5.3 Naive implementation

The naive/direct way to implement frontline reinforcement is to simply choose the supply route as a straight line. However this is not totally optimal. It could be the case that the distance between the back planet and the front-line planet could be too large. This means that if ships are sent directly then the fleet would be in air for a large amount of time and so we
cannot “change our mind” in the future turns when there is a change in our strategy.

This means that we want to send our ships to the front with a few stops along the way. These ideas appear in temp_dummy[5].

5.4 Algorithm

There have been a lot of suggestions as to what path to choose for this task. Some coders have in fact suggested even an A* search on our planets with an appropriate heuristic to find this path. However we have chose to keep it simple and choose the path in a greedy way as suggested in Khratsmov’s blog[1]:

Algorithm for Front Lining

For every planet P with free ships to spare on it after defence and offence moves:

1. Find the closest enemy E to this planet

2. If there is any of our planet F which lies in between P and E, then we have found a Frontline planet. We add it to list of all possible planets at the front line

3. Among all the planets at the frontline we choose the one which is closest to us(C) and also which is most into the front-line(L)

4. With a heuristic we decide if we should directly forward to L or pass it to C so that it eventually gets forwarded to L (or some other frontline planet)
The above figure shows the frontline planet for a planet at the back and also the shorter path and the longer path as calculated above for frontlining.

We modified the Khratsmov’s frontlining algorithm[\textsuperscript{1}] to account for the non-linear supply route. We chose an intuitive heuristic. However we didn’t get any efficient way to perform tests and decide the heuristic since the front-lining didn’t significantly change the game to observe the difference.

6 Heuristic Estimation

- At many points in our strategy, there were many possible actions to take, out of which it was possible to carry out only a few. Hence, we had to choose the best options among them. This required assigning each action a score/ heuristic and then ranking all the actions based on this score and carrying out the ones with the lowest heuristic value.

- We used the following general heuristic algorithm to determine the functional dependence of heuristic $H$ on factors $F_1, F_2, F_3, ..., F_n$: 
• \( H = ((F_1)^{f_1}) \ast ((F_2)^{f_2}) \ast \ldots \)

• If the action is not preferable on increase of a parameter \( F_i \) then \( f_i < 0 \), else it is \( > 0 \).

• We initially assume \(|f_i|\) to be 1 for all \( i \).

• The different factors used in the estimation of heuristic could influence the chance of taking the action or not differently. That is some factors may have higher influence on the action decision compared to others. We identify such factors and try to get the exponent for these factors.

We use the following methods to determine \( f_i \) for the above found factors:

• We choose two values \([l, r]\) between which \( f_i \) might lie. We construct two bots with exact same code except for the difference of \( f_i \). We fix \( f_i = l \) for one bot and \( f_i = r \) for other bot.

• We run game against both of the above bots for reasonable amount of maps, and take that choice of \( f_i \) which gives the wins in max number of games.

• We do an approximate binary search on the interval \([l, r]\) as described above, till we don’t see much changes in the number of wins.

7 Our Heuristics

7.1 Attack moves strategy

For every planet of ours with spare ships on it, we had to choose the best planet which is not ours to attack. For this we chose the heuristic:

For attacking neutral planets,
\[
\text{heuristic} = \text{DistanceToTarget} \ast (\text{numShipsToSend}^{A_1} / (\text{GrowthRateOfTarget}))
\]

for attacking neutral planets,
\[
\text{heuristic} = \text{DistanceToTarget} \ast (\text{numShipsToSend}^{A_2} / (\text{GrowthRateOfTarget}))
\]

where \( A_1 = 1.0, A_2 = 0.9 \) (found using method explained in the previous section).

7.2 Front lining

For each of the planets that do not attack any planets in a given turn, we send their ships to a front line planet. Now, we could send the ships directly
to a front line planet or send them to a planet close to them, and through a series of such planets, finally to some front line planets. This will give us the freedom to reroute them in between, i.e., when they are at some other planet, if needed. But this has a trade off that the ships may reach the front line too late if this path through planets is longer. Hence, we need to chose which path to take to the front line, and this is done by comparing the two distances as:

\[
\text{if}(A_1 \times (\text{distance to front line planet } + \text{distance of front line planet to closest enemy}) \leq (\text{distance to closest planet of ours } + \text{distance from that planet to nearest enemy planet})) \text{ then we send directly to the front line planet else, we use the path as described above.}
\]

\[A_1 = 1.4\] (found using the method explained in the previous section). Note that we took the distance mentioned on the rhs above, rather than calculating the total distance over all the entire path, because it is a good enough estimate.

### 7.3 Defense moves strategy

For each of the defence moves required, we can only fulfil some, and we choose those according to the heuristic:

\[
\text{heuristic} = (\text{TurnsRemainingToDefend}) \times (\text{GrowthRateOfPlanetToBeDefended})
\]

### 8 Strong test cases

- Our bot beats a top 50 bot, in 11 maps out of 100, when there are a bunch of planets very close by.
- We present 6 such test cases where this effect can be seen in Figure 1.
- This is because, we gave a lot of emphasis on defence moves, and when planets are close by, defence gets easier. Since defending becomes easier, we can attack in more turns, and hence our bot gets to be more aggressive.

### 9 Weak test cases

- We do not consider nearby enemy planets which could attack our planet, when we plan attacks from that planet. Hence, in some cases the enemy planet can capture our planet, which has been weakened by sending out ships for another attack. This can be seen in Figure 2(a).
Figure 1: good test case screen shots
Our bot is quite defence based and gives preference to attack on neutrals rather than on enemies, usually. Hence in Figure 2(b), even though we have less ships and growth rate, we still waste our resources on attacking a neutral (using the circled fleet), rather than attacking an enemy.

It is more preferable to capture center planets before the enemy does, because it gives us a better frontline (brings us closer to enemy). Since we don’t give any preference to this capture, we lose if the enemy uses such a strategy (as above), or captures the centre planets before us. This is especially visible in maps that have clear distinction of center vs corner planets as in Figure 2(c).

10 Possible improvements for bad test cases

- We could check near-by enemy planets and account for the ships in them, before launching an attack from any of our planets. This might lead to bot being too defensive and leads to a resistance to capture even easy targets. So, a heuristic has to be in place even for this for bad test case #1.

- Make the bot behave differently when it is winning as compared to when it is losing for bad test case #2.

- To counter the bad test case #3, we could use a heuristic as introduced in stuart’s blog[6].
• When the initial planets are close by, implement a kamikaze attack as described in Zilog9\[4\].

11 Weakness of bot and future scope of work

• Attack moves and defence moves are isolated from each other. Sometimes it is better to attack even if it means to give up one of our planets(Such cases can be seen in the bad test cases section).

• We use up all the resources for defence and only attack using the remaining resources.

• In a given turn from one of our planets, only one planet can be attacked.

• Multi-planet attack (also called 'gang attack') is not possible from any of our planets.

• When the initial birth/spawn planets are very close-by, the bot doesn’t play optimally. The best thing to do when this happens is implement KamiKaze attack described in Zilog9\[4\].

• Implement A* path finding algorithm to implement supply route as described in temp_dummy[5].

• Assign score to every attack action as described in Mistmanov[1]. Carry out an action only if it is possible to achieve positive score.
References


