Planet Wars AI

Cyborgs

Sourabh Ghurye
130050001

Nikhil Vyas
130050023

Utkarsh Mall
130050037

Algorithm

Future Calculation Function

Given the fleet informations for all fleets of our planets and enemy planets, the function calculates future till a time parameter *'t_max'*. This function simulates the growth rate, fleet movement and calculates owner and number of ships at a time on all the planets. The function is useful in calculating where to send fleets and the size of fleet in the next move. This functions acts as evaluation function for a **1-ply minimax**.

Defending

In our strategy, defending always takes place before attacking or capturing neutral planet. If enemy is attacking on our planet, we calculate the number of ships required to reinforce the planet under attack. This is calculated by calc function.

```
ships_required = 0
break_done = false
for k in 1:my_planet.ships-def_param:
      add_to_fleet(my_planet,under_attack,k)
      calc()
      if planet_owner[under_attack][t_max] = Player 1:
            ships_required=k
            break_done = true
            break
if !break_done:
      ships_required=my_planet.ships-def_param
```

Also it is possible to defend using more than one planets. If the first planet is unable to save planet, it sends all its ship except some 'def_param' ships, which are always to be kept for stability of itself. Now the above algorithm is applied from next nearest planet.

Capturing and Attacking

After defending we capture/attack with remaining ships in the turn. In a single move only one of capturing or attacking takes place. If capture/attack is not possible then, none of it takes place. We use calc function for each fleet size and dest planet and check for maximum profit attack.

The algorithm finds score(profit) for each such move as follows

```
ships_required = 0
tmax_score=0
break_done = false
for k in 1:my_planet.ships-def_param:
      add_to_fleet(my_planet,dest_planet,k)
      calc()
      if planet_owner[dest_planet][t_max] = Player 1:
            ships_required=k
            tmax_score=future_ships[dest_planet][t_max]
            break_done = true
            break
if !break_done:
      ships_required=my_planet.ships-def_param

if tmax_score-ships_required > max_score
      max_score = tmax_score-ships_required
```

The above algorithm is iterated over all planets, and the one with max_score is chosen to be the attacked or captured.

Bringing Ships to Front

It is a good move to bring ships to the front(closer to enemy). It helps is attack.

This was accomplished by iterating over each planet (*source*) at the beginning of the turn, and

1. finding the enemy planet closest to the *source* (*closest enemy*)
2. finding another planet of ours, *target*, that was closest to the *closest enemy*.
3. If *target* did not exist, then the *source* was a front planet, and didn't have to send its ships anywhere.
4. If the *target* did exist, then the *source* had to send some of its free ships to the *target*.

Gradient Descent Over Parameters

To find optimal parameters *t_max, def_param,* we use gradient descent over these:

```
init_param=random vector
for 100 times:
      if wins with init_param > wins with all neigbour_params:
            return init_param
      else:
            init_params <- neigbour_param with maximum wins
```

Code is present in GradDesc.py and GradDesc2.py

Test Cases Where Algorithm Excels

5, 26, 43, 47, 61, 87

The Algorithm performance excels in those cases in which starting planets are nearer as this helps in creating an initial strong position due to Frontier formation.

Against enemies like rage bot etc who do not coordinate in between moves, our algorithm usually wins as we are easily able to take into account their fleets by future calculation.

Test Cases Where Algorithm Fails

6,17,35, 59,74

The Algorithm predicts the future but there is no functionality to reserve ships to be sent later due to this ships that should be sent after the attack of enemy planet on neutral planet, are sent before.

The Algorithm does not take into account the nearness on enemy planets to the attacked planet which causes ships being wasted by attacking on a planet near to enemy.