Artificial Intelligence Project Report
Project 1 :: Planet Wars

By Team Paprika:
Harsh Parmar (130050017)
Shouvanik Chakrabarti(130050072)
Krishna Harsha(130050076)

Considerations and Motivations for Algorithm:

The inittial considerations for our algorithm are as follows:

- ***Naive Notion of Value in a State:*** Victory in Planet Wars can be obtained through two means. Either, all the alotted turns for the game will complete, and our bot is left with the most ships on planets, or the opponent runs out of ships entirely at some point of the game. Thus the main measure of value of a certain game state in Planet Wars is the difference in the number of ships owned by ourselves and our opponent. This forms the first naive basis by which we compare different moves to determine which would be the most advantageous.
- ***Description of State:*** We have written a new class wrapper for the Planets which maintains an updated notion of the state of each planet in the game. We add functions to this wrapper to maintain a timeline for the planet which gives its projected state consisting of the projected number of planets as well as the projected state. We also added a function that would allow us to simulate its timeline when we sent .
- ***Collective Action:*** Once we decide to conquer a certain planet, we must come up with a method of doing this efficiently using our current distribution of planets. The ideas of efficiency include conquering the required planet as fast as possible while exposing our own planets to the minimum possible risk. We come up with a numerical notion of the 'gain' from attacking a certain planet using a certain strategy. We then use coordinate descent to reach an optimal distribution of the ships being sent amongst the planets we possess, by maximizing this numerical gain.
- ***Numerical Gain:*** The numerical gain is computed by trying to estimate the difference in the number of planets with our opponent after a certain number of turns that we choose to look forward for, given the fleets currently in play. This should ideally be done by simulating the state of every involved planet with and without the moves we are making. While we did write functions that could do this this resulted in far too many calculations to carry out in the allowed computaion time for each turn and led to our bot arbitrarily timing out when the computational load became too large. We thus had to come up with heuristics that would allow us to estimate the numeric gain to a reasonable degree of accuracy. Specifically the heuristics were meant to estimate whether

we could conquer and retain a planet and the gain we would eventually make from it given the fleets in play. When we conquer a planet that belongs to the enemy we have both pros and cons. The pros are that we gain a planet with a certain growth rate, and our opponent loses that planet, thus after a certain horizon, the gap in the number of ships between us and our opponent will increase by twice the growth rate times the number of turns we have possessed it by then. The ships lost to conquer the planet dont matter as our opponent loses the same number. The cons are that we are exposing our planets to risk and if we estimate that our planet will get conquered, we incur a loss of twice the growth rate of that planet times the turns the enemy has it by the horizon. When we conquer a neutral planet we lose a certain number of ships to conquer it. We make a gain of the growth rate of the planet times the number of turns for which we have it. The loss due to the exposure of our own planets to risk remains the same as in the case of conquering enemy planets.
The heuristic we use to simulate the state of the conquered planet with respect to the fleets currently in play, assumes that all the fleets that appear after the planet is first conquered, reach the planet a constant T turns after, and thus completes a simple approximate simulation. Note that until the planet is conquered our simulation is exact for our purposes. After experimentation and comparisons, we chose 3 as the value to use for T.

- ***Choosing moves to make:*** We initially compute the best strategy to conquer every planet, and then choose to conquer the planets for which the strategy yields the highest gain. We also choose the second best strategy if the planets involved in it are mutually exclusive with the ones for the first strategy. We do this so that our strategy does not become fixated on one planet. We do only two distinct strategies per turn so that our ships do not get too diffused in one turn, executing mediocre strategies as in our experience there are few really viable strategies available each turn. If other strategies are very lucrative they can always be executed the next turn.

After the initial considerations, we used our bot for a while and based on our observations we came up with various heuristics to enable our bot to respond to certain situations. The main motivation was that so far our bot was behaving completely on the assumption that the fleets currently in play would be the only ones that would be relevant until the time. Some heuristics arising out of the distribution of planets and ships currently on the map could help us to make some degree of prediction regarding the consequences of moves based on the enemies subsequent actions.

- ***Addition loss incurred in sending ships out of a planet:*** Here we consider two further cons of sending ships out of a planet, that are not included in the calculation of numerical gain.
The first of these is that if the planet from which we send the ships out has enemy ships close by, these enemy ships might be in a position to conquer our planet faster than they would be otherwise. In our experience, enemies usually tend to attack an opposing planet with fewer ships that is relatively close to it.

We look forward to the same horizon used for numerical gain and estimate the loss we might incur due to such an eventuality.
The second of these is the loss of a potential oppurtunity to conquer a nearby planet by sending ships away from a host. This is estimated by considered the three closest planets and computing the maximum value obtainable from conquering one of them in the future at the horizon. The difference in this value with the ships sent and without is the loss of potential. This prevents us from making a move for mediocre gain now when we could have made a very profitable move otherwise just a few turns down the line.

- ***Additional notion of lucrativeness of a planet based on geographical proximity to other planets:*** We noticed that our bot was playing badly in planets were there were valuable clusters in parts of the field. This was because certain planets provide strategic advantages after being conquered in terms of the potential to attack other planets, and how defensible the position is. Through some experimentation we came up with a heuristic that expresses the geographical value of the planet in terms of its proximity to friend and enemy planets.
- ***Notion of defensibility of a position:***We felt that it was too naive of us to consider a positions value assuming that after we conquer it, we would possess it until the horizon. In real play, the enemy is very likely to send out ships to retake positions. Depending on the projected number of ships present on the friend and enemy planets that are close to the target, and their relative proximity, we estimate the number of turns for which we can hold a planet and use this to refine the value added in conquering a planet. In particular, positions that are defensible in the long term are now more preferred.
- ***Chaining in the attack strategy:***When a planet does not make any moves in a particular turn and is also not close to any enemy planet, we come to the conclusion that the ships on the planet may not be readily available for offense. In that case after we make our normal plays, we select the ships on that planet that it can spare without getting conquered down the line, and send it to the nearest friendly planet that has a lower minimum distance from the enemy planets.

Final Algorithm:

- At the start of every turn create wrapper objects for every planet.
- Then update the state for every object based on current fleets in play, so that the timelines for all of the planets are available.
- Compute the best strategy to conquer any planet. Also list the numerical gains for such a strategy.
- Consider only strategies for which the numerical gain minus the heuristic risk of our planets getting conquered (due to ships being sent away from dangerous spots).

- Sort the strategies based on a linear combination of the numerical gain, the heuristic risk and potential loss, and the geographically based additional value of the planet we are going to conquer.
- Issue orders corresponding to the best strategy in this sorted list. If the second best strategy is mutually exclusive in terms of the planets involved, also issue orders for that one.
- Issue orders so that planets that are relatively isolated and from which no moves have recently been made, can send their ships to more active planets with greater oppurtunities for offense, following our chaining strategy.

Situations where the bot succeeds:

The bot acts very offensively in the beginning and captures a signifcant number of valuable neutral planets, and eventually starts conquering enemies as well. This usually means that the number of ships we possess rises quite steeply once we have the time to get a significant offense going.
This makes our bot very successful in maps where there is a well distributed selection of valuable planets close to our starting position. This ensures that after our initial offense, we possess a bunch of well defensible closely spaced planets with a significant number of ships. This coupled with our chaining system to move ships into planets better suited for offense as well as our high propensity to defend our own planets is usually sufficient for us to beat most bots in such situations.
Another situation that helps us to succeed is if our starting planet is significantly far from the enemies home planet, which makes the enemy unlikely to attack us after the initial offense. When we are far away from the enemy, the enemy might not consider it of so much value to attack us an this helps us set up our initial offense.

Situations where the bot fails drastically:

The bot fails drastically in two major situations. The first and most severe one exposes the flaws in the initial strategy of the bot. Due to the nature of our heuristics we tend to greedily seek to conquer planets in the beginning. This can be absolutely fatal in the case where the initial planets are very close together and our enemy directly attacks our home base. This is especially bad for otherwise 'stupid' bots, as they are the ones that choose to attack the enemies home base in the first turn, when it cannot be assured of any gain or even of conquering the planet. Due to the nature of our heuristics and gain functions we will consider attacking the home base to be a viable startegy only after that planet sends out some ships, but on a map where the other choices of planets to conquer are very far away, the damage is done by then, as we lose our homebase and end up with only one or two small fleets bound for relatively distant planets which can then easily be cleaned up by the enemy. This failing can be clearly observed in the match between our bot and the “Rage Bot” provided as part of the Planet Wars starter package on the provided map 26.
On some very diffused maps, even when we get our initial strong offense off, the nature of the terrain results in our ships getting distributed over a range of relatively small and distant planets. An enemy that is less ambitious in the beginning might stay

concentrated in a portion of the map thus making their planets more defensible in general, as well as allowing for fast and targetted offenses of our more vulnerable planets.

Possible means of aleviating the current shortcomings:

Some things that would probably make our bot better, but which we did not have the time to implement and test:

- The notion of defensibility of a position is still flawed as it makes very crude assumptions of when an enemy would choose to attack a planet. Ideally we would like to study common game strategies and make our bot responsive to that.
- The notion of the distribution of all our planets would ideally be much better, and this could help us to avoid getting too spread out and thus putting ourselves in disadvantageous positions ie. making the planet “map aware”. The distribution of all of our planets is something that should probably be factored directly into the value of our game state. Currently, we have anotion of this but it is on the basis of largely experimental heuristics that are factored in while sorting the quality of strategies.
- We would like to come up with a way to either simulate the state of a planet given a hypothetical attack more accurately. We did implement the code to exactly calculate this but it was too inefficient and ended up being unusable.