Assignment 2: Strategic AI

My freshman year, playing Galcon Fusion was a fairly common pastime between my roommate and I. Over that year we became fairly proficient at the game, always trying to beat each other's scores on a certain AI difficulty and even playing head-to-head on a regular basis. Using the skills I had gained from my freshman year, I tried to employ similar strategies of varying success.

When I played the game years back, my strategy typically included expanding to nearby planets as fast as possible, then fending off "snipe" attacks (as they've been called) by quickly reinforcing previous planets, all while throwing ships at the sniping planet, since they were oftentimes left weakened after sending out so many ships. While these properties appeared simple to implement on paper, I quickly ran into some issues.

AI Implementation 0.0:

Force the player to expand as fast as possible to neutral planets, using heuristics designed from metrics such as the target planet's growth rate and distance from the source planet and send 10% of its ships over. From here, implement a finite state machine that will switch between expand, reinforce, attack, and snipe depending on the state of the world.

This implementation was a disaster. I didn't have a system in place to take metrics on the algorithm at that time, but it was easy to tell that it failed against the majority of the bots on any of the maps I tested, especially RageBot. Before I could even get to the finite state machine, it became obvious that finding reliable heuristics for planet selection was no basic task. I very briefly debated writing some sort of genetic algorithm to determine these heuristics, but I was afraid the implementation might have already been doomed from the start. Forcing the player to quickly expand at the beginning of the game left them especially vulnerable to implementations that stayed safe and garnered resources at the beginning of the round instead of wasting their starting ships capturing nearby planets.

I honestly didn't expect to stick with this implementation, but I hadn't expected it to fail so horribly either. What I was trying to avoid was diving straight into a MiniMax implementation that would eventually have to be scaled down to prevent the player from timing out. After the implementation, however it did become evident that I needed some tool to look into the future and determine advantageous moves.

It was then I began constructing my World class, which I could manipulate in any way I felt fit. The class essentially worked as a model for an instance of PlanetWars. I could add potential fleets to the model and step the model n steps in the future. A basic scoring function allowed me to keep track of which models produced the most favorable future result. The simple action of stepping forward in time also gave the AI a way to react to the last actions sent by their opponent.

AI Implementation 0.1:

Have each planet individually simulate sending fleets of 20%, 40%, … 100% to each other planet and look 30 turns ahead on each fraction to determine which action or lack thereof would produce the highest future score. The action is then issued.

Bot	*Wins*	*Losses*	*Draws*
Bully	*100*	*0*	*0*
Dual	*100*	*0*	*0*
Prospector	*100*	*0*	*0*
Rage	*95*	*5*	*0*
Random	*100*	*0*	*0*

The flaws with this implementation on paper seem fairly obvious. For starters, because each planet acts independently, it doesn't consider that there may be other planets better suited to complete the task it set itself out to complete. As an example, one planet, *A,* may consider that it needs to reinforce another planet, *C,* all the way across the board from an incoming snipe attack. However, planet *B* is directly adjacent to *C* and has an abundance of resources to protect the planet. With the current implementation, both planets *A* and *B* would send their ships to *C* if it those actions were determined to be the most advantageous to the player instead of just sending *C*'s ships and sparing planet *A*'s resources to use for another task.

Despite these inherent flaws in the idea, the algorithm performed exceptionally well, losing only 5 games out of 500 against the sample AI. I tested these all using a program I later made to try my AI iterations against the bots. It took some tweaking of the fraction values and future steps to get the results above, but what's interesting is that this algorithm was never intended to be a serious option and actually came about as a simple way for me to test that my World model was behaving correctly. Due to its success, however, I decided to use it as a basis for my algorithm. The player never came close to timing out and appeared to be operating intelligently in the replays.

From here I looked into ways to defeat RageBot in *all* cases. I tried a number of strategies, such as waiting out the first few turns and moving only once we could determine what type of adversary we were up against, specifically tracking how many ships in how many fleets they sent out over a period of time. This strategy, however, only put the player at a disadvantage against bots who took advantage of the first few turns and resulted in a bunch of draws with RageBot, since the player often chose to stay put and absorb the brunt of RageBot's attacks. The implementation would have likely been weak against non-static AI implementations in either case.

I also tried reinforcing allied planets close to enemies, but couldn't manage to do this effectively enough without leaving other planets susceptible to snipe attacks from the enemy. After seeing the theories tested by our classmates, I believe an influence map could have been useful here, but at the time of designing this strategy, I honestly hadn't considered using one. At the time, I was getting increasingly worried that my "greedy"

expansion design didn't actually use any AI formalisms, especially without the inclusion of a FSM.

After all the tinkering, I took a closer look at the maps where my bot performed the worst: 22, 26, 27, 30, and 61. What I noticed on these maps was that both players' starting planets started extraordinarily close to one another:

Map	Distance
22	5
26	6
27	11
30	5
61	7

(It's important to note here that the average starting distance between the two planets was 19). Using this information, I speculated that having the player behave more cautiously in these special cases might prevent getting shutdown early by aggressive bots.

[Aside: Alongside these tests, I was also trying to improve how quickly I could win the games I was already winning in order to prevent the enemy from finding intelligent ways to come back from a near-defeat. All of my implementations for this case were extremely risky, such as only targeting enemy planets when I felt I was well enough ahead, which led to a large number of preventable losses. Since I determined I was better off carrying on intelligently as before, I decided to scrap the idea.]

AI Implementation 0.2:

If the home planets start within 10 steps of each other, perform a variation of Minimax, dubbed "Rage Defend", to prevent against rage attacks. "Rage Defend" ends once the player gains more resources than the opponent. After "Rage Defend" ends, perform AI Version 0.1, dubbed "Greedy Search", for the remainder of the matchup, instead scoring by total ships on fleets and planets as opposed to -just- total ships on planets.

Bot	*Wins*	*Losses*	*Draws*
Bully	*100*	*0*	*0*
Dual	*100*	*0*	*0*
Prospector	*100*	*0*	*0*
Rage	*96*	*1*	*3*
Random	*100*	*0*	*0*

After entertaining the thought long enough, I finally caved and decided to implement a version of Minimax to avoid early rage attacks from the opponent. This way, I could ensure optimal decision-making at the beginning stages of the game before acting greedily in order to prevent timeouts in future turns. With this model, I finally managed

to establish a small FSM with "Rage Defend" and "Greedy Search" states in addition to a more legitimate AI algorithm.

In order to get the most out of my Minimax, I started the first iteration off as inclusive as possible. For my turns, I simulated sending fleets in groups of $0\%, 10\%, \dots, 100\%$ from each of my planets and fleets in groups of $0\%, 20\%, \dots, 100\%$ from each of the enemy's planets for each of their turns. For every terminating node, I looked 30 steps in the future to determine the states' scores. Rage Defend, as it became called, was only terminated once the player gained an advantage over their opponent, assuming that they would have acted optimally up until that point.

This implementation failed pretty miserably. Unsurprisingly, it's difficult to simulate $\left((p_{player} * p_{total} * 10) * (p_{enemy} * p_{total} * 5)\right)^{steps}$ nodes 30 steps into the future in under a second.

From here, I went straight on to implementing Alpha-Beta pruning and seriously restricting the number of possibilities considered. I limited the possible fractions of ships sent on my end to increments of $1/5$ and the enemy to $1/3$, limited the number of evaluated steps to 1, and set the terminal nodes to look only 10 steps into the future. The latter two decisions were made under the assumption that Rage Defend would only have to run at the beginning stages of the game where only few planets will have been captured, if any at all and most fleets are sent to planets equal to or within 10 steps away, since any further would leave the enemy susceptible to counterattack.

After the values were tweaked, the algorithm worked exactly as intended, as can be seen in the results above. The player managed to win one more game on RageBot and managed to force a draw on another 3, where presumably the most optimal strategy was not to move until the enemy made a mistake. The player did still lose one game to RageBot on map 27, in which case Rage Defend was never run because the players began at a distance >= 10 away. Changing the starting range of Rage Defend seemed dangerous since a larger allowed distance could potentially cause the bot to timeout. Because testing the bot against on all maps takes a considerable amount of time, other, more drastic, design options were considered before testing variations of this value.

Although Implementation 0.2 is the design I submitted for this assignment, I managed to perform a series of other implementations, afterwards. One such design involved treating each planet as their own entity that could switch between Rage Defend and Greedy Search depending on their proximity to the nearest enemy planet, but this design was quickly shutdown. In many cases, planets on the frontline would become too timid and not help nearby planets suffering from enemy attacks, resulting in a near 50% winrate per bot, a value approximated from a series of manual tests.

Another idea I tested was only checking future steps up until turn 200. Essentially, simulated steps would not exceed turn 200 as a way of more accurately simulating how the game would end up. The hope was that the bots would become more risky and deploy last-ditch efforts to secure points or close out games, but the results and average number of turns taken for wins remained essentially unchanged.

Previous iterations of the design as well as some intermediate steps between AI iterations can be found online at:

https://github.com/JGMEYER/JustianMeyer-CS4731-Project2