Spike: Task 7

Title: PlanetWars Tactical Analysis

Author: Sam Huffer, 101633177

Goals / Deliverables

- Multiple bot agents for the PlanetWars simulation, including at least one that uses tactical analysis to inform its decisions.
- Numerical comparison of the bots' performances over multiple maps.

Technologies, Tools, and Resources Used

- Command prompt (for executing and testing the code)
- Visual Studio (VS) 2017 (for editing code)
- Learning materials on Canvas (for instructions and sample code)

Tasks Undertaken

- I copied the PlanetWars project from last week into the week 7 folder, and replaced the existing bots with the bots Naïve, SimpleTactical, and ComplexTactical.
- For Naïve's decision making, having it attack at random was the only way I could think of to have it attack without using any external information, so I had it use a modified version of the logic used by Rando from last week, making it randomly choose from the lists neutral_planets, enemy_planets, and not_my_planets, and then choose a random destination from the selected list if it's not empty, using not_my_planets if it is.
- For SimpleTactical's decision making, I had it switching between three states based on planet availabilities:
    - Attacking (target enemy's high-production planets) when the enemy has 5 or less planets and SimpleTactical has at least 5 more than them or the condition for Growing wasn't met.
    - Growing (targeting any high-production planet) when the condition for Attacking wasn't met and the enemy had more planets.
    - Waiting (not sending new fleets) when there are no unowned planets available to attack.
- For ComplexTactical's decision making, I took SimpleTactical's code and further built upon it, with three additional states:
    - Defending (reinforcing a high-production planet) if ComplexTactical has at least 10 planets and one of its best producers' defences were below average, or if the enemy had launched a fleet attacking a high-production planet.
    - Raiding (target a newly undefended planet) when a new fleet is detected and ComplexTactical would otherwise be Attacking or Growing.
    - Sabotaging (damage a planet's defences without capturing it) when it determines the actions for Attacking or Growing would not be viable.
- To make ComplexTactical's fleet detection work, I made a custom EventDispatcher class that Naïve, SimpleTactical and ComplexTactical all used to throw events whenever they dispatched a new fleet. Whenever ComplexTactical spotted these events, it noted the source and destination of the fleet to better determine a course of action.

- To run the bot comparisons, I modified the code in main.py's main procedure to loop through a game a preset number of times, printing to the terminal the outcome of each game and the running total of each bot's wins, losses and draws. When running the bot performance comparisons, I selected three random maps with a d100 (6, 26 and 88). For each map, I made the bots play 20 matches, recorded the results, then swapped their player numbers (and therefore starting planets) and made them play another 20 matches.

Code Snippets

```
from random import choice
from pyglet.event import EventDispatcher

class BotEvents(EventDispatcher):
    def __init__(self):
        self.register_event_type('on_new_fleet')

    def new_fleet(self, src, dest):
        self.dispatch_event('on_new_fleet', src, dest)
```

Figure 1: The custom event dispatcher for declaring the creation of new fleets.

```
# check if we should attack
if gameinfo.my_planets and gameinfo.not_my_planets:
    self.check_state(gameinfo)
    src = None
    dest = None
    dests = []

    #Attacking: getting enemy's highly productive planets
    if self.state is State.Attacking:
        print("Attacking")
        dests = self.rank_by_productive(gameinfo.enemy_planets.values())

        for p in dests:
            src = self.find_closest_to(self.find_all_stronger(gameinfo.my_planets.values(), p.num_ships + (2 * self.leave_in_reserve)), p)

            if src is not None:
                dest = p
                break

        if src is not None:
            print("Found source")
            self.dispatch_fleet(gameinfo, src, dest, int(src.num_ships - self.leave_in_reserve))
        else:
            print("No known suitable source")

    #Growing: getting any highly productive planet, enemy or not
    elif self.state is State.Growing:
        print("Growing")
        dests = self.rank_by_productive(gameinfo.not_my_planets.values())

        for p in dests:
            src = self.find_closest_to(self.find_all_stronger(gameinfo.my_planets.values(), p.num_ships + (2 * self.leave_in_reserve)), p)

            if src is not None:
                dest = p
                break

        if src is not None:
            print("Found source")
            self.dispatch_fleet(gameinfo, src, dest, int(src.num_ships - self.leave_in_reserve))
        else:
            print("No known suitable source")

    #Waiting: "And neutral jing, when you do nothing." - King Bumi
    elif self.state is State.Waiting:
        print("Waiting...")

    #Other: an error has occured
    else:
        print("Error: AI ComplexTactical.state has not been asigned a valid value")
        return
```

Figure 2: The actions taken by SimpleTactical when assessing what to do in each state.

```
#Attacking: getting enemy's highly productive planets
def attack(self, gameinfo):...

#Defending: reinforcing my most productive planets
def defend(self, gameinfo):
    print("Reinforcing high-production planets")

    if self.defend_enemy_target():
        dest = self.new_enemy_fleet.dest

        if dest not in self.best_planets:
            self.best_planets.append(dest)
    else:
        dest = self.find_weakest(self.best_planets)

    src = self.find_strongest_excluding_best(gameinfo.my_planets.values(), self.best_planets)
    self.dispatch_fleet(gameinfo, src, dest, int(src.num_ships / 2))

#Growing: getting any highly productive planet, enemy or not
def grow(self, gameinfo):...

#Raiding: attacking a planet the enemy just left vulnerable
def raid(self, gameinfo):
    print("RAIDING!!!")
    stronger_on_combat = self.find_all_stronger_on_combat(gameinfo.my_planets.values(),
                                                          self.new_enemy_fleet.src,
                                                          self.new_enemy_fleet.src.num_ships + (2 * self.leave_in_reserve))

    src = self.find_closest_to(stronger_on_combat, self.new_enemy_fleet.src)

    if src is not None:
        self.dispatch_fleet(gameinfo, src, self.new_enemy_fleet.src, int(src.num_ships - self.leave_in_reserve))
    else:
        self.grow(gameinfo)

#Sabotaging: attacking an average-production enemy planet from a low-production planet with lots of ships
def sabotage(self, gameinfo):
    print("Nope, sabotaging instead . . .")
    src = self.find_least_productive_but_strongest(gameinfo.my_planets.values())
    dest = self.find_median_production(gameinfo.enemy_planets.values())
    self.dispatch_fleet(gameinfo, src, dest, int(src.num_ships / 2))
```

Figure 3: The code for the additional states in ComplexTactical

Bot Comparisons

Table 1: The results of matches on maps 6, 26, and 88 between Naïve bot and SimpleTactical bot, 20 matches per position per map, max game length 2000.

Map	Bot	Player no.	Wins	Losses	Draws
6	Naïve	1	0	20	0
	SimpleTactical	2	20	0	0
	Naïve	2	3	17	0
	SimpleTactical	1	17	3	0
26	Naïve	1	1	19	0
	SimpleTactical	2	19	1	0
	Naïve	2	0	20	0
	SimpleTactical	1	20	0	0
88	Naïve	1	0	20	0
	SimpleTactical	2	20	0	0
	Naïve	2	1	19	0
	SimpleTactical	1	19	1	0

Table 2: The results of matches on maps 6, 26, and 88 between Naïve bot and ComplexTactical bot, 20 matches per position per map, max game length 2000.

Map	Bot	Player no.	Wins	Losses	Draws
6	Naïve	1	4	15	1
	ComplexTactical	2	15	4	1
	Naïve	2	2	18	0
	ComplexTactical	1	18	2	0
26	Naïve	1	0	20	0
	ComplexTactical	2	20	0	0
	Naïve	2	1	19	0
	ComplexTactical	1	19	1	0
88	Naïve	1	2	18	0
	ComplexTactical	2	18	2	0
	Naïve	2	0	20	0
	ComplexTactical	1	20	0	0

Table 3: The results of matches on maps 6, 26, and 88 between SimpleTactical bot and ComplexTactical bot, 20 matches per position per map, max game length 2000.

Map	Bot	Player no.	Wins	Losses	Draws
6	SimpleTactical	1	20	0	0
	ComplexTactical	2	0	20	0
	SimpleTactical	2	20	0	0
	ComplexTactical	1	0	20	0
26	SimpleTactical	1	20	0	0
	ComplexTactical	2	0	20	0
	SimpleTactical	2	20	0	0
	ComplexTactical	1	0	20	0
88	SimpleTactical	1	0	20	0
	ComplexTactical	2	20	0	0
	SimpleTactical	2	20	0	0
	ComplexTactical	1	0	20	0

Table 4: The overall results for each matchup over both positions across all three maps.

Bot	Wins	Bot	Wins	Draws
Naive	5 (4.2%)	SimpleTactical	115 (95.8%)	0 (0%)
Naive	9 (7.5%)	ComplexTactical	110 (91.7%)	1 (0.8%)
SimpleTactical	100 (83.3%)	ComplexTactical	20 (16.7%)	0 (0%)

What I Found Out

Both SimpleTactical and ComplexTactical flattened the Naïve bot, with a 95.8% and 91.7% win-rate respectively across all three maps. When facing each other, however, SimpleTactical won every scenario 20 games to 0 except for when it was player 1 on map 88, where the opposite scenario occurred. I suspect that its generally higher win-rate against Naïve was because its code was much more aggressive than ComplexTactical's and focused solely on attacking high-production planets, only differentiating between which set of planets to choose a target from. The only reason I can think of

that ComplexTactical won on map 88 as player 2 would be that that map afforded player 2 a great enough positional advantage that ComplexTactical could overcome the higher-production advantage that SimpleTactical's more aggressive strategy afforded it.

Extension Considerations

Additional Tactical Information

Consider what additional "tactical" information could be analysed and exploited.

Information that was analysed by SimpleTactical and ComplexTactical:

- The no. of planets each player controls.
- No. of ships at each planet.
- Production rate of each planet.
- Distance between planets.
- Source and destination planets of a fleet (via that information being put into an event).

Information that could be analysed and exploited:

- The no. of ships in a fleet.
- Possibly, which planets are visible or not to the enemy's fog of war, if bots could be made aware of that.

Fog of War Implications

The PlanetWars simulation supports a "fog of war" view of the game environment, where each bot agent only has partial (incomplete, possibly incorrect) information about the current state of the game. Explore what the implications of this are, and how they could be exploited.

Human players could likely make an educated guess of which planets were or were not inside their enemy's field of view and thus concealed by the fog of war. Using such information, they could build up ships just outside of their opponents' awareness and maximise their chances of conducting an impactful surprise attack, perhaps even from seemingly random locations if they position their ships correctly. I would imagine that it would be possible to program bots to use such tactics. Tricky for students perhaps, but possible.

Implications of Asymmetrical Maps

Most of the game maps provided are symmetrical. What does an asymmetrical map create in terms of game bias (game balance) as well as tactical opportunity?

If a map is symmetrical in its distribution of planets, then the players / bots have equal opportunity to capture well positioned or high-production planets, and the result of the match would be determined based solely on the skill of the player or the sophistication of the bot's algorithm. That is not the case with asymmetrical maps as, by definition, one player has access to more high-production planets or is closer to a larger number of planets than the other, giving them an advantage. I'd argue this was evidenced by the games played between SimpleTactical and ComplexTactical, specifically on map 88: on each other map tested, SimpleTactical won all games, but on map 88, the bot that was player 2 won all games through a suspected tactical advantage over player 1 great enough to overcome the strategic inferiority of ComplexTactical's algorithm when compared to SimpleTactical.