Spike: 10
Title: Tactical Analysis with PlanetWars
Author: Luke Valentino, 103024456

Goals / deliverables:
Use tactical analysis to influence agent decisions in the context of a strategic game model.

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work

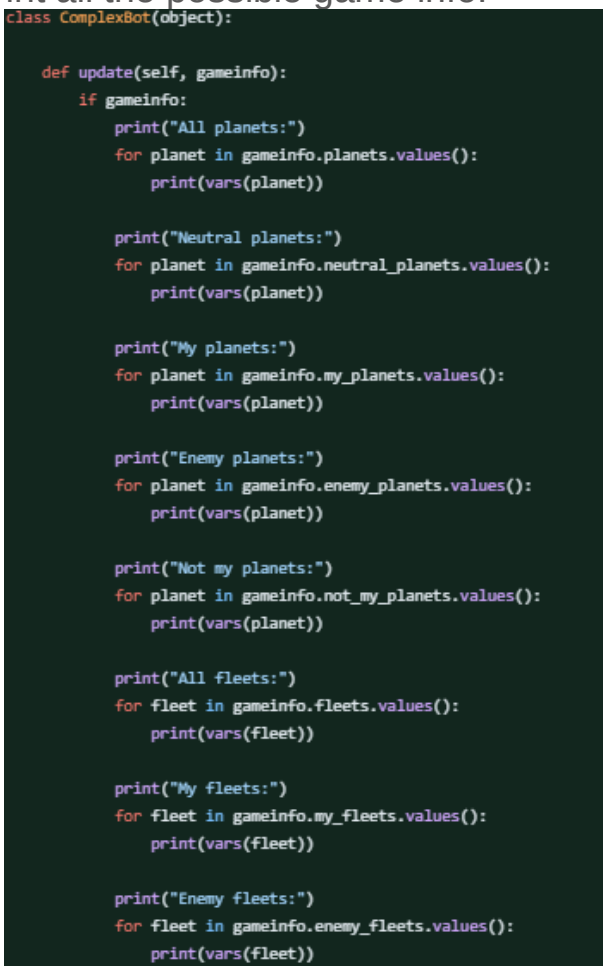
- Cloned Planet Wars Code
- Swinburne Lecture Notes

Tasks undertaken:
After importing the previous Planet Wars code, we must create a new bot file. This will be our ComplexBot.

- Now that our new bot has been created, we have to add it to the simulation.
    - In the settings portion of the bottom of the main.py file, edit the players to look like this.

```
'players': [
    #'OneMove',
    #'Blanko',
    'Rando',
    #'Simple',
    'ComplexBot'
    #'PingPong',
],
```

- Then to get a better understanding of the tactics available to use, we can use the ComplexBot to print all the possible game info.



```
class ComplexBot(object):

    def update(self, gameinfo):
        if gameinfo:
            print("All planets:")
            for planet in gameinfo.planets.values():
                print(vars(planet))

            print("Neutral planets:")
            for planet in gameinfo.neutral_planets.values():
                print(vars(planet))

            print("My planets:")
            for planet in gameinfo.my_planets.values():
                print(vars(planet))

            print("Enemy planets:")
            for planet in gameinfo.enemy_planets.values():
                print(vars(planet))

            print("Not my planets:")
            for planet in gameinfo.not_my_planets.values():
                print(vars(planet))

            print("All fleets:")
            for fleet in gameinfo.fleets.values():
                print(vars(fleet))

            print("My fleets:")
            for fleet in gameinfo.my_fleets.values():
                print(vars(fleet))

            print("Enemy fleets:")
            for fleet in gameinfo.enemy_fleets.values():
                print(vars(fleet))
```

- Now that we will be able to see our available intel we can create tactics. The tactical setup I implemented was as follows.

My ComplexBot uses two distinct tactics: scouting and attacking.

Scouting Tactic:
- Objective: Get information on neutral and enemy planets
- Execution:
  - For each bot-controlled planet with more than 10 ships, identify targets to scout.
  - A target is a neutral or enemy planet that is prioritized by their vision_age and proximity.
  - If a targets vision_age is more than 5, send scouts.

```
    def scouting_mode(self, gameinfo, my_planets, neutral_planets, enemy_planets):
        for planet in my_planets:
            if planet.num_ships > 10:
                # Identify targets that need scouting
                scout_targets = sorted(neutral_planets + enemy_planets, key=lambda p:
(p.vision_age, self.distance(planet, p)))[:3]
                for target in scout_targets:
                    if target.vision_age > 5:
                        # Send scouts using fleet_order
                        num_scouts = min(5, planet.num_ships - 10)
                        if num_scouts > 0:
                            gameinfo.fleet_order(planet, target, num_scouts)
```

Attacking Tactic
- Objective: Send ships to control neutral and enemy planets.
- Execution:
  - Create a target list of enemy and neutral planets, sorted by their growth_rate and number of ships from most to least.
  - For each controlled planet, if it has more than 1.5 times the ships of a target, send a fleet.
  - Attack fleets consist of 75% of a planet's ships.

```
    def attack_mode(self, gameinfo, my_planets, enemy_planets, neutral_planets):
        targets = sorted(enemy_planets + neutral_planets, key=lambda p: (p.growth_rate,
p.num_ships), reverse=True)
        for planet in my_planets:
            for target in targets:
                if planet.num_ships > target.num_ships * 1.5:
                    # Send attack fleet using planet_order
                    num_ships_to_send = int(planet.num_ships * 0.75)
                    gameinfo.planet_order(planet, target, num_ships_to_send)
                    break
```

What we found out:

The tasks and strategies used in this spike relate to multiple ULO's:

1. Discuss and Implement software development techniques to support the create of AI behaviour in games:
   - The creation of ComplexBot required the planning and analysis of AI behaviours included AI vs AI behaviour.
4. Create agents that are capable of planning actions in order to achieve goals
   - The scouting part of ComplexBot in particular demonstrates this ULO. By using a tactic to gather more information, the bot can better determine the best course of action.

Complex Bots Stats

	Games Played	Games Won	Win%	Average Steps	Longest Game (steps)	Shortest Game (steps)
Map 5	20	20	100%	127	233	102
Map 10	20	20	100%	112	190	72
Map 25	20	20	100%	134	289	69
Map 30	20	20	100%	60	97	20
Map 55	20	20	100%	76	105	39

These are the stats from ComplexBots games. In every game he was facing 1 other bot, Rando.py. As can be seen from the data, the outcome of each game was the same even when using different maps.