Spike: Spike Week 10
Title: Tactical Analysis with PlanetWars
Author: Nguyen Khanh Toan - 104180605

Goals / deliverables:
Develop at least two distinct "bot" agents for the PlanetWars simulation.
• One bot can be a basic bot previously created, ensuring it makes valid moves.
• Another bot must utilize tactical analysis to guide its decision-making process.
• In this spike report, clearly explain the tactical analysis incorporated with bot's strategy with a relevant excerpt of the bot's code within the spike report.
• Compare the performance of each bot numerically across various maps and multiple runs to mitigate random variations in results. Present the performance outcomes in your spike report, preferably in a table format, though visual representations like charts are recommended.
Examples of Tactical Bot Strategies:
• Simple: Prioritize targets based on factors such as "weakest," "strongest," "closest," or highest productivity planets.
• Advanced: Implement event detection (e.g., identifying vulnerable planets), scouting or fog-of-war manipulation, defensive or aggressive modes, and scouting strategies.

Technologies, Tools, and Resources used:
- Visual Studio Code
- Python 3.12

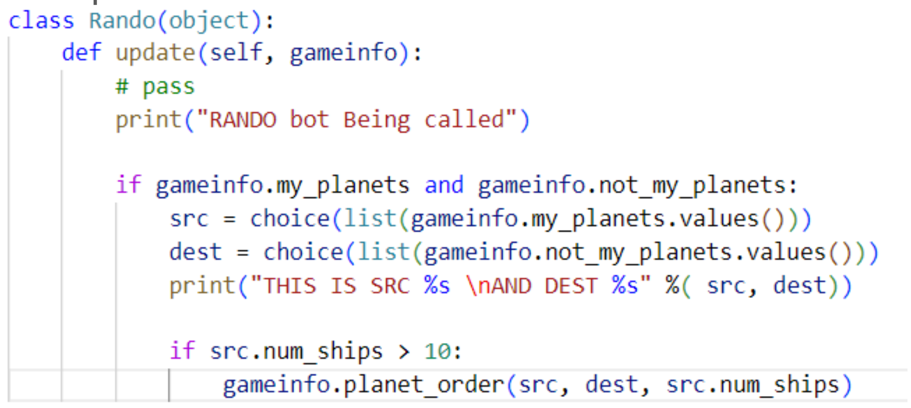
Tasks undertaken:
- Install Python 3+
- Install and setup compatible IDE for the language, e.g.: Visual Studio Code
- Pay attention to the comment of how the code work and functionality. Can use debug tool to observe the program more clearly.
- Run the code and observing the output.

Planning Notes (Include Extension planning):
- Starting with the game environment and add more variables to indicate a planet state, for instances health, shield, power of ships, …
- Implementing the GOAP to control actions coupling with goals of the agents.

- Add more random state of the environment such as “Fog”, “Black hole”, obstacles that affect the AI agents’ decision making… This require agent to adapt their plan to the environment.
- Add more goal and actions to AI agents to make the agent more complex.
- Different type of agents with varying strategies and functionality.

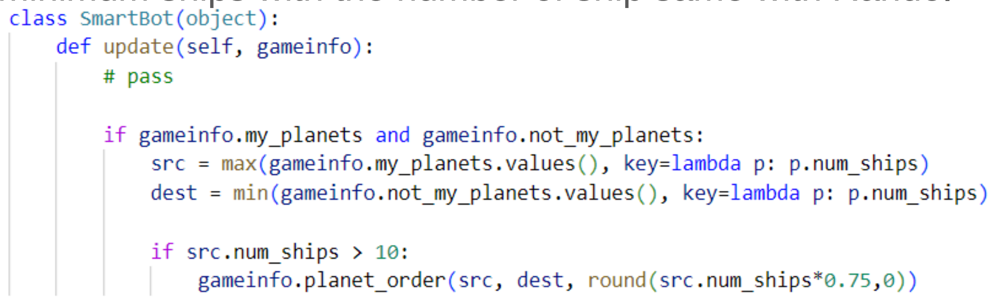
What we found out:

- The simulation containing two Bots, which is Rando and Smart Bot in Rando.py and SmartBot.py respectively.
- Rando Bot will randomly generate random source and destination planet, but only choose 75% ships of the planet that having more than 10 ships



```
class Rando(object):
    def update(self, gameinfo):
        # pass
        print("RANDO bot Being called")

        if gameinfo.my_planets and gameinfo.not_my_planets:
            src = choice(list(gameinfo.my_planets.values()))
            dest = choice(list(gameinfo.not_my_planets.values()))
            print("THIS IS SRC %s \nAND DEST %s" %( src, dest))

            if src.num_ships > 10:
                gameinfo.planet_order(src, dest, src.num_ships)
```

- Smart Bot will generate source and destination planet by choosing maximum ships of the planet and send it to the planet that having minimum ships with the number of ship same with Rando.



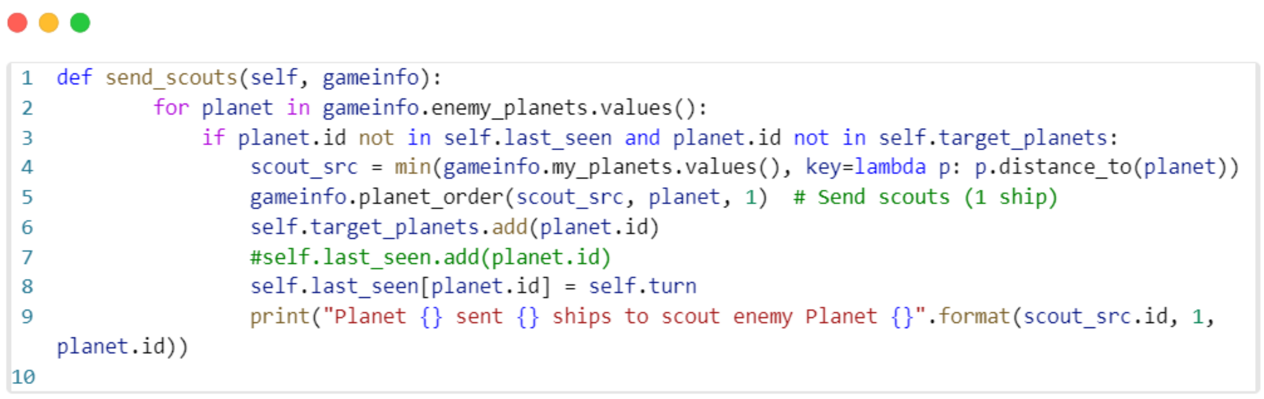
```
class SmartBot(object):
    def update(self, gameinfo):
        # pass

        if gameinfo.my_planets and gameinfo.not_my_planets:
            src = max(gameinfo.my_planets.values(), key=lambda p: p.num_ships)
            dest = min(gameinfo.not_my_planets.values(), key=lambda p: p.num_ships)

            if src.num_ships > 10:
                gameinfo.planet_order(src, dest, round(src.num_ships*0.75,0))
```

- Disciplarmy Bot, this is an bot with an improvement in determining action for extension.
- For the A.I strategy for PlanetWars game, Disciplarmy Bot contains a set of rules to scout, defend and attack.
  • Tactical analysis: Disciplarmy Bot considers the vulnerability and strategic
  importance of planets when attacking and defending.
  • Fog Of War: Disciplarmy Bot manages information by tracking the last

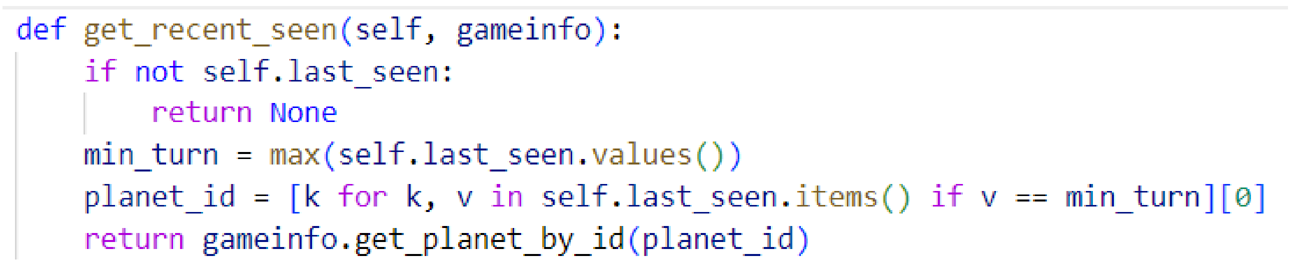
seen state of enemy plannets and updating its knowledge with new scouting information.

• The order of sequence of this bot is choosing to send 1 ship to scout, and then prioritising to defend its week planet then attack.

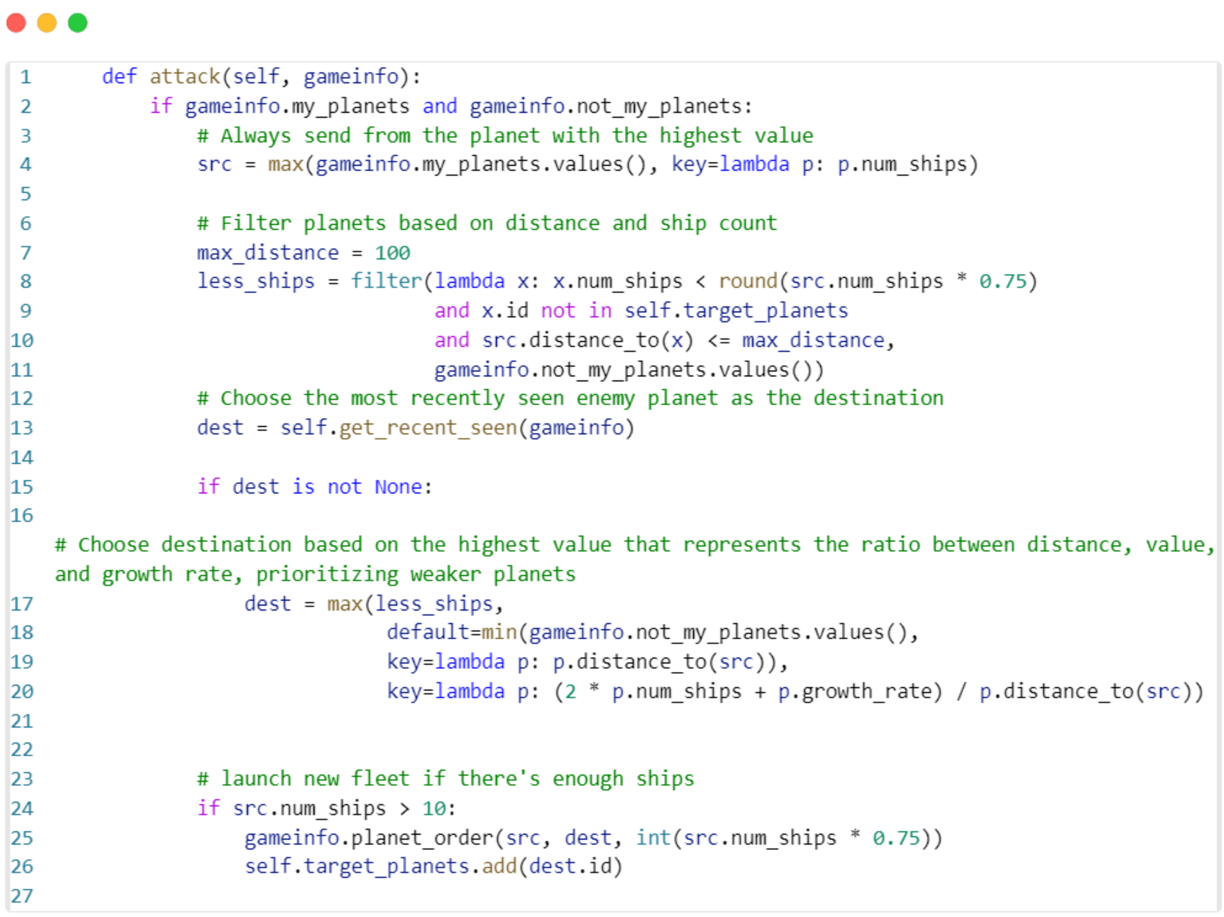
• The couting function will store last planet it discorverd in last_seen set() and attack() will use get_recent_seen() to find the last planet that discorved to attack by let 75% of ships from the planet have maximum ships.



```python
def send_scouts(self, gameinfo):
        for planet in gameinfo.enemy_planets.values():
            if planet.id not in self.last_seen and planet.id not in self.target_planets:
                scout_src = min(gameinfo.my_planets.values(), key=lambda p: p.distance_to(planet))
                gameinfo.planet_order(scout_src, planet, 1)  # Send scouts (1 ship)
                self.target_planets.add(planet.id)
                #self.last_seen.add(planet.id)
                self.last_seen[planet.id] = self.turn
                print("Planet {} sent {} ships to scout enemy Planet {}".format(scout_src.id, 1, planet.id))

```

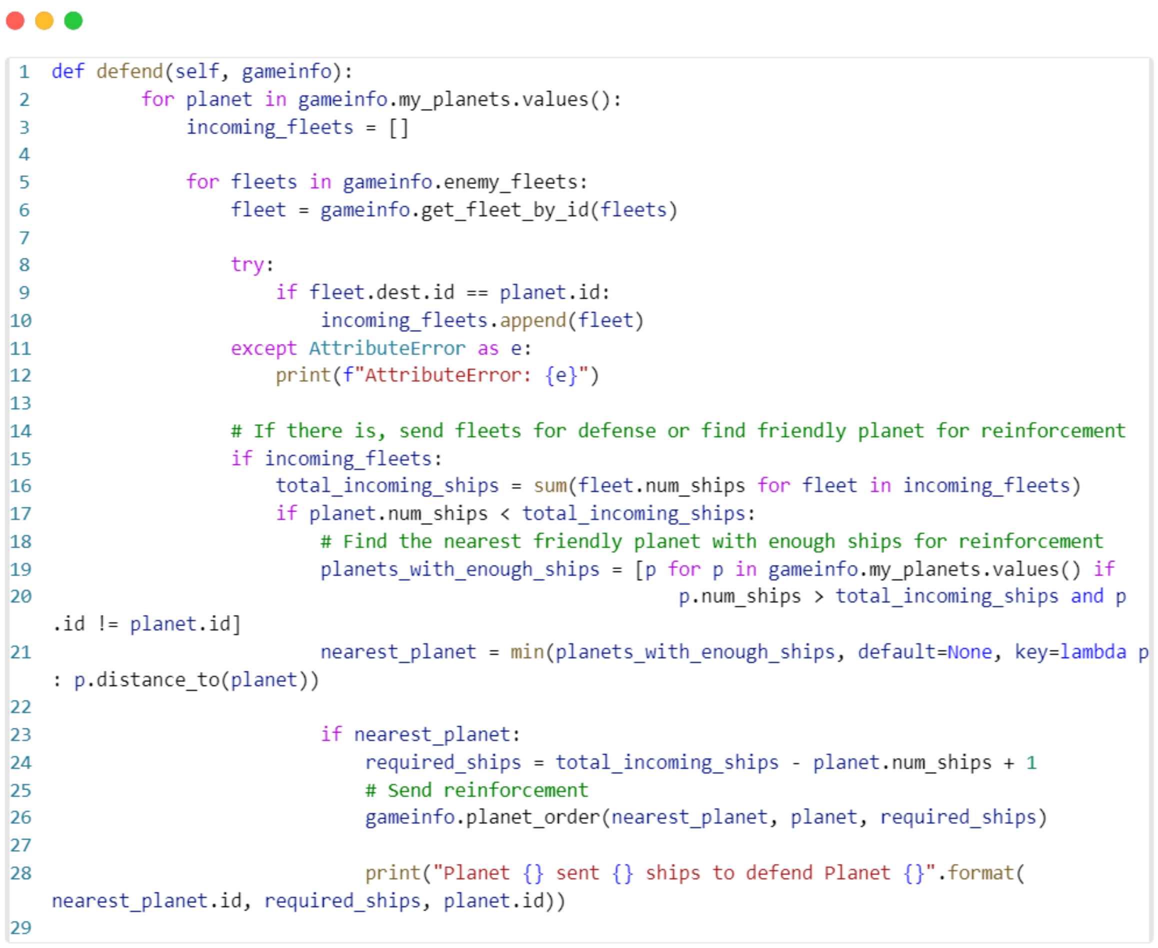


```python
def get_recent_seen(self, gameinfo):
    if not self.last_seen:
        return None
    min_turn = max(self.last_seen.values())
    planet_id = [k for k, v in self.last_seen.items() if v == min_turn][0]
    return gameinfo.get_planet_by_id(planet_id)
```

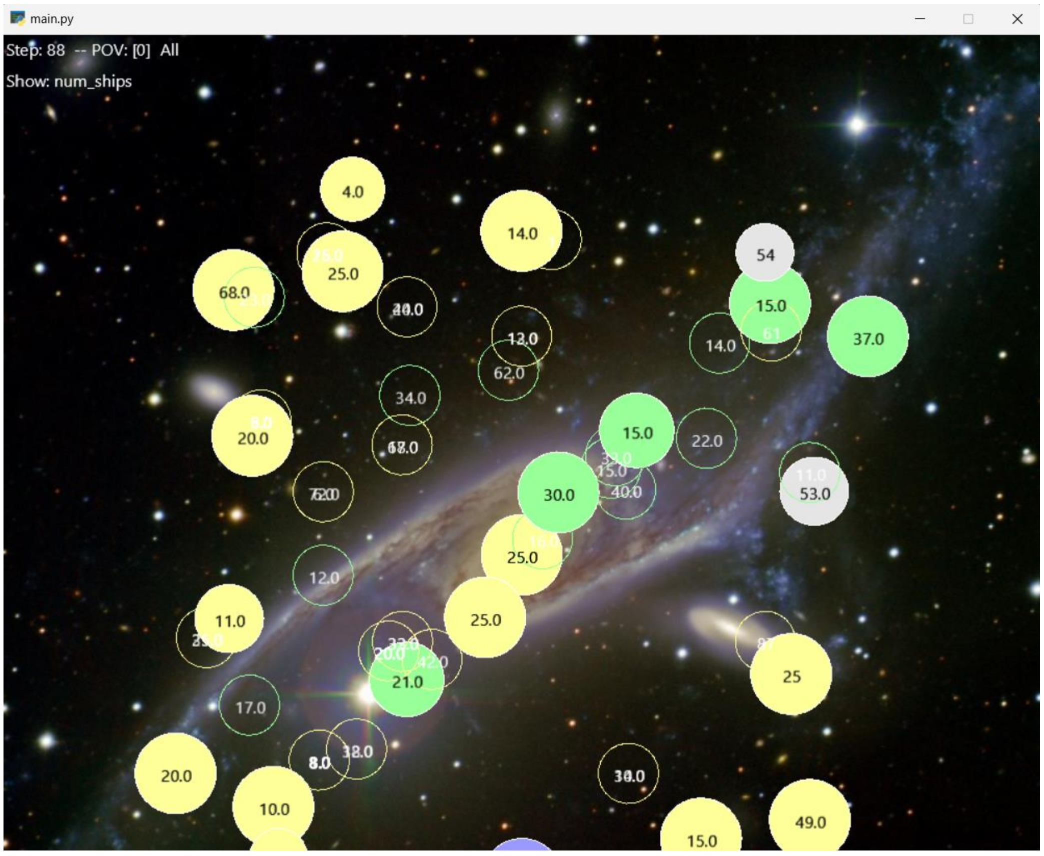


```python
    def attack(self, gameinfo):
        if gameinfo.my_planets and gameinfo.not_my_planets:
            # Always send from the planet with the highest value
            src = max(gameinfo.my_planets.values(), key=lambda p: p.num_ships)

            # Filter planets based on distance and ship count
            max_distance = 100
            less_ships = filter(lambda x: x.num_ships < round(src.num_ships * 0.75)
                                    and x.id not in self.target_planets
                                    and src.distance_to(x) <= max_distance,
                                    gameinfo.not_my_planets.values())
            # Choose the most recently seen enemy planet as the destination
            dest = self.get_recent_seen(gameinfo)

            if dest is not None:

# Choose destination based on the highest value that represents the ratio between distance, value, and growth rate, prioritizing weaker planets
                dest = max(less_ships,
                            default=min(gameinfo.not_my_planets.values(),
                            key=lambda p: p.distance_to(src)),
                            key=lambda p: (2 * p.num_ships + p.growth_rate) / p.distance_to(src))


            # launch new fleet if there's enough ships
            if src.num_ships > 10:
                gameinfo.planet_order(src, dest, int(src.num_ships * 0.75))
                self.target_planets.add(dest.id)

```

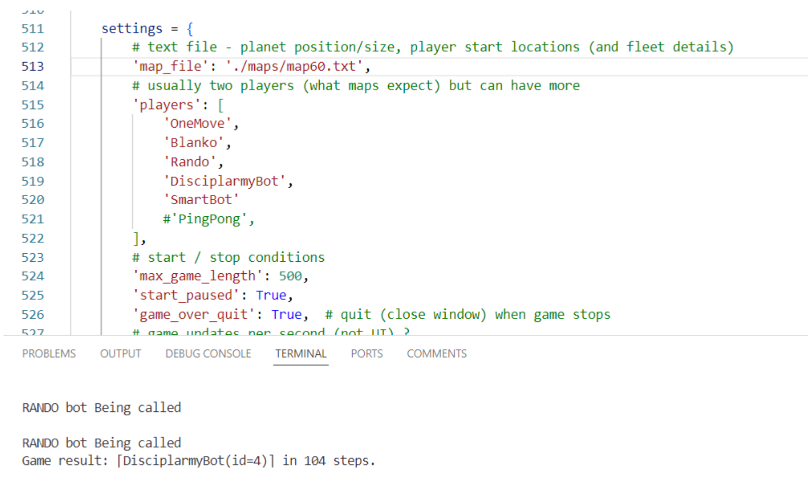
• The defend() will track the number of ship from enemy planet in the world and send army to reinforce allies.



```python
def defend(self, gameinfo):
        for planet in gameinfo.my_planets.values():
            incoming_fleets = []

            for fleets in gameinfo.enemy_fleets:
                fleet = gameinfo.get_fleet_by_id(fleets)

                try:
                    if fleet.dest.id == planet.id:
                        incoming_fleets.append(fleet)
                except AttributeError as e:
                    print(f"AttributeError: {e}")

                # If there is, send fleets for defense or find friendly planet for reinforcement
                if incoming_fleets:
                    total_incoming_ships = sum(fleet.num_ships for fleet in incoming_fleets)
                    if planet.num_ships < total_incoming_ships:
                        # Find the nearest friendly planet with enough ships for reinforcement
                        planets_with_enough_ships = [p for p in gameinfo.my_planets.values() if
                                                        p.num_ships > total_incoming_ships and p.id != planet.id]
                        nearest_planet = min(planets_with_enough_ships, default=None, key=lambda p: p.distance_to(planet))

                        if nearest_planet:
                            required_ships = total_incoming_ships - planet.num_ships + 1
                            # Send reinforcement
                            gameinfo.planet_order(nearest_planet, planet, required_ships)

                            print("Planet {} sent {} ships to defend Planet {}".format(nearest_planet.id, required_ships, planet.id))
```

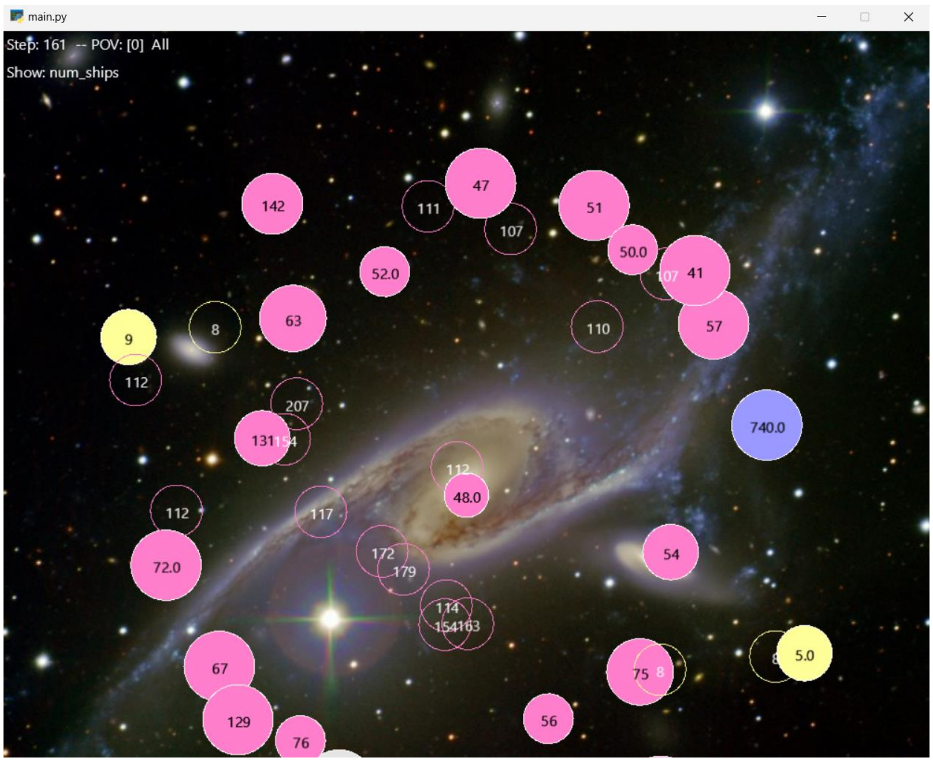


Output 1: Disciplamy Bot Win (Map 30)

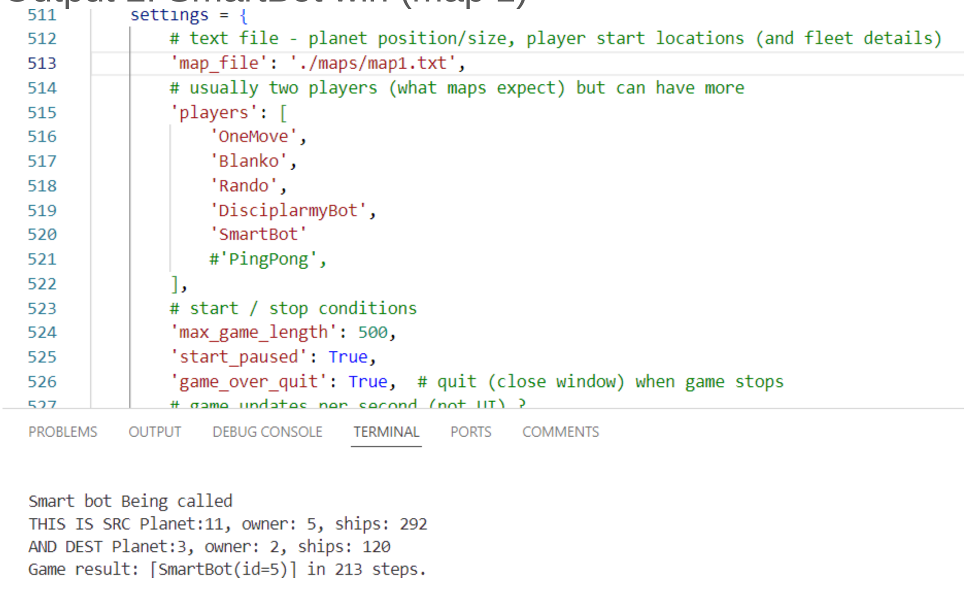


```
    settings = {
        # text file - planet position/size, player start locations (and fleet details)
        'map_file': './maps/map60.txt',
        # usually two players (what maps expect) but can have more
        'players': [
            'OneMove',
            'Blanko',
            'Rando',
            'DisciplarmyBot',
            'SmartBot'
            #'PingPong',
        ],
        # start / stop conditions
        'max_game_length': 500,
        'start_paused': True,
        'game_over_quit': True,  # quit (close window) when game stops
```

```
RANDO bot Being called

RANDO bot Being called
Game result: [DisciplarmyBot(id=4)] in 104 steps.
```



Output 2: SmartBot win (map 1)



```
    settings = {
        # text file - planet position/size, player start locations (and fleet details)
        'map_file': './maps/map1.txt',
        # usually two players (what maps expect) but can have more
        'players': [
            'OneMove',
            'Blanko',
            'Rando',
            'DisciplarmyBot',
            'SmartBot'
            #'PingPong',
        ],
        # start / stop conditions
        'max_game_length': 500,
        'start_paused': True,
        'game_over_quit': True,  # quit (close window) when game stops
```

```
Smart bot Being called
THIS IS SRC Planet:11, owner: 5, ships: 292
AND DEST Planet:3, owner: 2, ships: 120
Game result: [SmartBot(id=5)] in 213 steps.
```