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

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

# Tasks Undertaken

* 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,
* For ComplexTactical’s decision making,
* 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 used a d100 to select three random maps to use, and rolled 6, 26 and 88. For each map, I had the bots play 20 matches, then swapped their player numbers (and therefore starting planets) and play another 20 matches. At the end of each set of 20 matches, I recorded the number of wins, losses and draws for each bot.

# 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 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| Naïve | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |
| 26 | Naïve | 1 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| Naïve | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |
| 88 | Naïve | 1 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| Naïve | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |

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 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| SimpleTactical | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |
| 26 | SimpleTactical | 1 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| SimpleTactical | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |
| 88 | SimpleTactical | 1 |  |  |  |
| ComplexTactical | 2 |  |  |  |
| SimpleTactical | 2 |  |  |  |
| ComplexTactical | 1 |  |  |  |

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 |
| Naive |  | ComplexTactical |  |  |
| SimpleTactical |  | ComplexTactical |  |  |

# What I Found Out

If the intention of the code provided to students was that we should modify it to address all the addi­tional functionality listed in the comments and have the program reach a “Done” state:

* When only one value needed to be evaluated (the effect on the goal being worked towards), the program worked perfectly fine and reached a “Done” state within a few loops.
* When evaluating the intended effect *and* side effects of an action, it was dependent on the magnitude of all the effects of the actions as to whether the program got stuck in a loop where reducing to one goal to 0 required the other goal to be addressed, and so on.
* If multiple goals are to be completed through GOB and SGI in a manner based upon that used in this spike, there need to be actions that will satisfy one goal and not negatively impact others; this is not a concern if the AI is not expected to reach a “Done” state where all goals are completed and the program ends.

With regard to the creation of the object-oriented version of the code:

* Object-oriented programming encourages the division of fields and methods into more man­ageable segments that themselves can be passed around the code to access their encapsu­lated data or asked to perform tasks expected of them.
* Object-oriented programs can take advantage of inheritance and polymorphism such that dif­ferent AIs have the same methods and fields but they’re implemented / populated differently, allowing for a variety of behaviours when the same actions of each sibling class are called by others.
* One downside of object-oriented python code specifically is the requirement of specifying self.[FIELD NAME] or self.[METHOD NAME] when a class accesses its own fields or methods, something that feels tiresome when other languages, such as C#, don’t require it as it is assumed that the field / method belongs to the class calling it unless otherwise specified.