a. Jerry Wu, wu000169

b. My strategy involves the creation of “border” planets, which are planets that have at least one edge that leads to an enemy that I own. From there on, I can have parameters that feed to each border planet to look at the state of the planets next to it. If the planet is empty for example, the borderplanet will send half its population to it. I also implement the whole list of planets as a graph using JGraphT in a weighted graph, then I used Dijkstra’s algorithm to create a dictionary of the shortest path between each planet and every other planet. This is used to make sure planets that are not border planets themselves and are also not directly next to border planets are able efficiently get to a border planet and supply it with population.

c. I used sets, dictionary, list, and map in my program. Sets was necessary to ensure no duplicate commands were given out, as all the potential .offer()s are first put into a set, and offered at once together at the end of the turn. They were also used to store the different types of visible planets such as conquered and unconquered and bordered. The dictionary, list, and map are part of the shortest path algorithms, where the dictionary was used to store the starting point as the key, and another dictionary which contained the target point and list of steps to target as the value.

d. Only strategy of project4.jar

e. I used JGraphT in my program to create the graph of the planets

f. Not the most complex AI, but should still win 70% of the time for AI 1 and AI 2