# Project 4 - Planet Wars

Csci 1933, Sect 1, Lab Sect 2

Professor Chad Myers

Don Le (lexxx675) & Zhongyi Sun (sunx0725)

First, we loop through the list of planets and sort them by visible planets we own versus visible planets that are neutral or owned by the opponent; the two categories of visible planets are stored in separate HashMaps. We start by looping over the list of neutral/opponent planets. For each planet, we should get the planets that are connected to it. If the current planet is neutral, we should send people from one of our connected planet to it in order to maximize overall population growth while avoiding abandoning a planet. If the planet is an opponent's, we should see if any of our connected planets has enough people to successfully attack the planet. If none of our planets has enough people alone, then check if we can use a combined attack based on the projected population, using the getProjectedPopulation method, and depending on the average distance. After looping over neutral/opponent planets, we loop over the planets that we own and, again, get the planets that are connected to each of these. If we find a planet that is 'safe' i.e. only surrounded by allied planets, we should check to see if we should send reinforcements to neighbors. Two PriorityQueues were used to store the projected populations of the surrounding, ally planets depending on their distance from the current planet. The two PriorityQueues were separated by whether the planet was projected to be owned by oneself versus opponent owned or neutral. If we are not projected to lose any planets, then we send people to the planet with the lowest projected population using peek() on the queues. If, instead, we are projected to lose planets, we should send reinforcements to a planet we're losing using peek() again.

## HashMap

HashMaps provide a convenient way of associating something for a particular planet with its ID as the IDs are unique, and we also can easily access them given the IDs. Thus, HashMaps were used for associating IDs with planets themselves, their distances from planets, and subsets of planets, like those connected to a planet of interest. This info, like shortest distance, can be used to optimize which planets we should send shuttles to.

## List

We used lists to keep the IDs for planets. When we want to get planets or information about those planets, we can loop through the list of IDs and use them as keys to get the related information because we stored planets in a HashMaps (info such as, planet, distance, and so on) .

## Queue - PriorityQueue

PriorityQueues store the projected populations based on distance to the destination (from the previous HashMaps). The data in these PriorityQueues will be naturally ordered, and we can use the fact that its first element is the smallest number, which we can access this  by calling peek (). Using this characteristic provides a simple way to understand which planets should be higher priority and should have shuttles sent to.

One issue found is that sometimes planets will continue to grow to be very large without sending reinforcements to neighboring planets appropriately; this means we are not using our resources efficiently and it can cause problems on graphs where the bridge between one side and the other consists of two connected planets. Because reinforcements are not sent properly, the game can easily reach a stalemate.