Project 1

Part 1:

1. We've implemented a roadmap search for this project.
2. Initially, we identify the nearest asteroid and generate random vertices in the environment into a list along with ship's position.
3. Then we generate edges between these vertices within a specific distance and see there are no obstructions in between for the edge to form and finally generate a graph and show graphics in the environment.
4. BFS and DFS algorithms are applied to traverse for target (asteroid) in the graph.
5. After identifying the path, the ship moves to the designated goal vertex.
6. Replanning is done when the location of the target asteroid is not present and also after some timesteps to not make the path longer to travel.

Part 2:

- generateRandomVertices(): This will generate the random vertices in a space
- generateEdges(): Constructs edges between vertices, provided there are no obstructions in the path.
- setShipAndGoalVertex(): Places the ship at the initial vertex of the list and sets the target as the final vertex in the list.
- pathGraphics(): Utilized to render the visual representation of the ship's path, be it via BFS or DFS.
- targetNotPresent(): This function determines if the specified target location ispresent, used in the dynamic replanning of the graph path.
- The Graph class holds lists of all vertices and their potential connecting edges.
- The Vertex class details the vertex position and identifies if it's a start or end node.
- The Edge class provides information on the pair of vertices forming that edge.
- The Search class describes a vertex and its connecting edge to another vertex.
- In general, the objective in the graph's path is to target asteroids for resource collection. While in the process if the ship's energy drops below 2000, it will seek out and move to the nearest beacon. If the collected resources exceed 800, the ship will return to the base.

Part 3:

We took the Co-operative mode which will focus on gathering resources, so that here we'll know what we are competing against in the environment.

Testing the Clients

To implement BFS (Breadth-First Search) or DFS (Depth-First Search), consider the following modifications in the line 137 and 138 of 'Gnan58Client.java' file code.

```
            Asteroid asteroid = pickHighestValueNearestFreeAsteroid(space, ship);
            constructGraph(space, ship, currentPosition, asteroid.getPosition(), graph);
            BreadthFirstSearch bfs = new BreadthFirstSearch();
//          DepthFirstSearch bfs = new DepthFirstSearch();
            path = bfs.traverse(allVertices.get(0));
```