Jacob Biederman

The first architecture choice I made was to implement DijkstraPathfinder and AStarPathfinder as children of GridPathfinder, and to use the existing grid and nodes rather than creating another layer of inheritance. I did try to create a DijkstraNode that inherited from node to store a back pointer and cost. However, this turned out to be hard to implement correctly without implementing a corresponding grid, so I just added the extra variables to the base Node class. The downside of this is that each node reserves unnecessary memory in both Dijkstra and DFS. I also chose to use the STL Priority Queue in both Dijkstra and A\* for the open list because it provides a fast way to access the next lowest cost node. Since I needed to search through the open list I inherited from Priority Queue and added a find operator, I also added an end operator to mirror vector operations. It might have been better for find to just return a boolean rather than an iterator.

If I had infinite time I would:

* fix the grid costs for extra credit. This would be a fairly easy fix and maybe take an extra half hour to an hour.
* Find a cleaner way to store the additional node variables. Inheritance is one option that could work, or implementing an additional struct or class to hold node information based on node ID.
* Fix mysterious bugs with pathfinding. Occasionally one click would lead to two pathfinding searches while debugging and occasionally the back pointers on the path would not work. To handle this, I just checked to make sure the path is valid, instead of fixing an intermittent bug without clear reproduction steps.