The Algorithm

To start off, we chose to set a determinate path from the source to the sink that takes exactly 6 edge placements to achieve. We then subtract 6 from the randomly chosen number for the edge pile between 15 and 30. We then augment the graph in the following way:

- Each addition to the graph ALWAYS happens in pairs, meaning we don't add one edge at a time, but a pair of edges instead.
- The algorithm iterates from a master list of vertices in the graph to determine what the next two edges will be. This is done through a **greedy** algorithm by scoring the edge pair based on total maximum flow between them. This is calculated by taking the minimum of the two max flows between each individual edge and storing it as it's "score."
- The way the first edge is chosen from the pair is that it checks for all legal moves from the current visited vertex, and stores in a temporary array all the ones that lead to a vertex **not** currently already in the graph. Then, the second added edge only considers all the legal moves that end up in a vertex that **does** exist in the graph. This is to make sure that the resulting graph is still connected, and no hanging edges are placed.
- Finally, the algorithm scoring prefers new edge pairs that originate from the source or end in the sink, to make sure that more viable paths are added to the graph. Also, the score of the added pair of edges is dramatically increased if the adding them to the graph causes augmentation of the flow.
- The graph is redrawn after each pair of edges is placed, and the max flow is re-computed. To simplify the process, since the edges are added in pairs, if the amount of edges is odd, then the last edge simply gets ignored.
- Once each pair of edges is placed, the graph does not change its mind, since a greedy method was used to begin with, rather than brute force.

To compute the flow, we used the Edmonds-Karp variation that used Breadth-First Search on the graph to validate paths from the source to the sink.