1. (20%) Create a design **before** you start coding that describes or shows how a graph structure could be used to store some kinds of data and attempt to solve some kind of problem (yes, this can be a game that needs a graph to represent a map!)

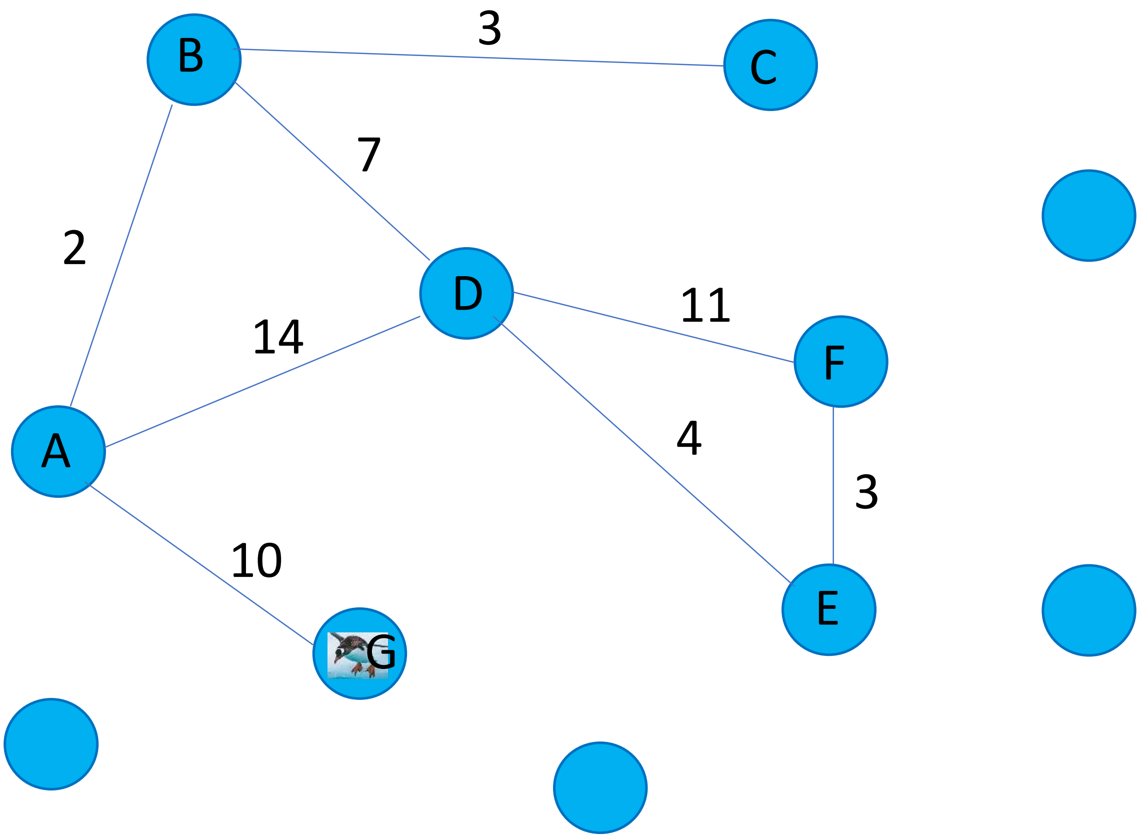
   Graph Structure for Penguin Routes:

   - **Vertices**: Icebergs with fun names.
   - **Edges**: Ocean paths connecting icebergs.
     - Bi-directional penguins can travel either way.
   - **Weight**: Length of the path (distance between icebergs).
     - Used for MSP and shortest path

   Problem to Solve:

   - Add graphs and vertices as new currents and icebergs are found.
   - Find the shortest path for penguins to travel from one iceberg to another for optimal fishing spots. Finding the path with the minimum total distance between two icebergs.
     - Dijkstra's
     - https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
     - Wikipedia: Dijkstra's algorithm
   - Determine the minimum spanning tree to establish efficient routes for penguins to explore new fishing spots while covering all icebergs.
     - Kruskal's
     - https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-using-stl-in-c/

Example of set up:



2. (20%) Create some tests (at least *two* for each piece of functionality) **before** you start coding...

3. (40%) Implement a graph class with at least (this category effectively combines implementation and specification, partly to emphasize getting the algorithms working!):

    1. (5%) a function to add a new vertex to the graph (perhaps add_vertex(vertex_name)),

    2. (5%) a function to add a new edge between two vertices of the graph (perhaps add_edge(source, destination) or source.add_edge(destination)),

    3. (15%) a function for a shortest path algorithm (perhaps shortest_path(source, destination)),

    4. (15%) a function for a minimum spanning tree algorithm (example min_span_tree()).

4. (10%) Analyze the complexity of all of your graph behaviors (effectively a part of our documentation for grading purposes),

5. (10%) Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).