Coders Inc Design Decisions – Milestone 2

# Distance Algorithms

* Decision was made to remove the base calculation for the distance between two values from the kNN function
  + This allows changes to be made to how the distance is calculated for different data types without having to change the kNN function
  + This also allows for easier testing of distance to help to isolate any bugs that may be present in either the distance or kNN functions

# Sole Distance Metric of Euclidean Distance

* The only function for calculating the net distance between two points that is present in the program is an Euclidean distance function
* Due to other issues and trouble tracking down troublesome bugs, the decision was made to forego the implementation of other distance functions for this Milestone
  + There are plans to include options for the user to pick from following distance metrics, on top of Euclidean
    - Minkowski Distance
    - Chebyshev Distance
    - Manhattan Distance

# Standardization using Normalization

* The decision was made to use normalization in order to standardize the points
* This is based on the assumption that the information that the user provides will approximately follow a normal distribution
  + We felt that this was a reasonable assumption to make based on the characteristics of normal distributions

# Use of Composite Design Pattern for Point Attributes

* The composite design pattern was used for the contents of a Point’s attribute
  + This allows the attributes to take simple, single-value attributes, as well as complex, multi-value attributes (such as a set of coordinates)

# Use of MVC pattern for GUI

* The GUI was built on the Model-View-Controller model.
  + The View is solely for the look of the GUI, and is responsible for responding to the controller classes and when the model is updated by setting up components.
  + The Model class is the already existing DimensionalSpace class. A method was added that updates the View when a point is added to it
  + The Controller classes are responsible for receiving user input and deciding what to do with it. It makes changes to both the Model and the View.
  + Multiple controller classes were used to keep classes concise and specific to certain situations, and to avoid having to check the source of the ActionEvent
* This pattern allows distinct communication between classes, and decouples the responsibilities of the physical view of the GUI and the look & feel of the GUI

# Use of MainController Interface

* Every controller has the interface MainController
* This interface was used so that the Model (of type DimensionalSpace) does not have to passed around and updated constantly