Ron Richardson

DATA 520 – Intro to Programming

Project 1

**Introduction**

ProTracer is an application built in Python to visualize the path and trajectory of golf shots of professional golfers using data collected by the PGA Tour ShotLink Intelligence System. This application can visualize the shots in either 2D or 3D for one or more shots at a time.

**How the Data was Collected**

The data was collected by the PGA Tour every year since 2003. In the specific trajectory data file, a device called TrackMan is used. TrackMan collects data using a camera and doppler radar from the time of impact all the way until the ball lands. Included in the data file are properties such as:

* Ball speed
* Club head speed
* Apex height
* Vertical launch angle
* Time of flight
* X, Y, and Z coordinates of the ball flight

**Building the GUI**

The GUI was built using QtDesigner, which is an application that comes bundled with Anaconda. QtDesigner allows the user to draw out their GUI using drag and drop tools and built in components. This allows for quick prototyping of a GUI and minimal coding. There are two ways to load this GUI into the Python application. First, the saved UI file from QtDesigner can be loaded in from the file system and rendered on the fly as the application is running. The second method, which I chose, was to compile the UI file into Python code from the command line, using the command pyuic5. This converts the UI file into Python code, which can then be modified, if needed, within the Python code rather than having to go back to QtDesigner and make changes there.

**Visualizing the Shots**

Since the data contains X, Y, and Z coordinates, it is easy to plot this using matplotlib. The tricky part came when I tried to embed the plot within the GUI itself. When running matplotlib code from IDLE or any other command line type interface, the plot will open up in a new window will the full navigational controls and mouse interactions (zooming, rotating, etc.). I found this to not be possible when embedded in the GUI, as the method of plotting the visualization is different. This is due to the mouse\_init() function not being called again once the axes are cleared. I was not able to get this working, despite a few online resources saying it might be possible, just by calling mouse\_init() manually.

In order to embed the plot in the GUI, I had to use a Figure and FigureCanvas object, which are part of the matplotlib.backends.backend\_qt5agg module. Once these widgets are added to the GUI, as shown below, the plot is then drawn onto the canvas. Additionally, it is necessary to clear the axes each time an iteration of the animation is performed, and then redrawn again from the beginning value up to the current iteration value.

class PlotCanvas(FigureCanvas):  
 def \_\_init\_\_(self, parent=None, width=8, height=6, dpi=100):  
 fig = Figure(figsize=(width, height), dpi=dpi)  
 self.axes = fig.add\_subplot(111)  
  
 FigureCanvas.\_\_init\_\_(self, fig)  
 self.setParent(parent)  
  
 FigureCanvas.setSizePolicy(self,  
 QtWidgets.QSizePolicy.Expanding,  
 QtWidgets.QSizePolicy.Expanding)  
 FigureCanvas.updateGeometry(self)

def update\_2d\_lines(self, num, datas, lines):  
 self.canvas.ax.clear()  
 for i in range(len(self.data)):  
 x = self.data[i][0, :num]  
 y = self.data[i][2, :num]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, linewidth=self.linewidth, label=label)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()  
  
def plot\_2d(self):  
 self.xmax += self.padding  
 self.ymax += self.padding  
 self.zmax += self.padding  
 self.aspect\_ratio = self.xmax // float(self.zmax)  
  
 self.canvas.figure = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 self.canvas.ax = plt.axes(xlim=(0, self.xmax), ylim=(0, self.zmax))  
 self.canvas.ax.get\_xaxis().set\_visible(False)  
 self.canvas.ax.get\_yaxis().set\_visible(False)  
  
 longest = -1  
 for i in range(len(self.data)):  
 longest = max(longest, self.data[i].shape[1])  
  
 ani = animation.FuncAnimation(self.canvas.figure, self.update\_2d\_lines, longest, fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False, init\_func=self.init\_2d)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()

The above code was pieced together from several different questions/answers on StackOverflow. I was disappointed to learn that the navigation toolbar was not available for 3D visualizations when embedded in a GUI. This takes away the mouse interactions which make 3D visualizations unique, in my opinion.

**Sources Used**

* <http://pyqt.sourceforge.net/Docs/PyQt5/index.html>
* <https://pythonspot.com/en/pyqt5/>
* <https://pythonspot.com/en/pyqt5-matplotlib/>
* <https://stackoverflow.com/questions/43947318/plotting-matplotlib-figure-inside-qwidget-using-qt-designer-form-and-pyqt5>
* <https://stackoverflow.com/questions/36222998/drawing-in-a-matplotlib-widget-in-qtdesigner>
* <https://stackoverflow.com/questions/12459811/how-to-embed-matplotlib-in-pyqt-for-dummies>
* <https://stackoverflow.com/questions/3972158/how-to-plot-on-my-gui>
* <https://stackoverflow.com/questions/36665850/matplotlib-animation-inside-your-own-pyqt4-gui>
* <https://stackoverflow.com/questions/4899176/qt4-mplot3d-of-matplotlib>
* <https://stackoverflow.com/questions/29357442/example-of-embedding-matplotlib-in-pyqt5>
* <https://stackoverflow.com/questions/42983449/python-getting-a-matplotlib-figure-to-rotate-when-embedded-in-a-gui>
* <https://stackoverflow.com/questions/41167196/using-matplotlib-3d-axes-how-to-drag-two-axes-at-once>
* <https://stackoverflow.com/questions/30330912/rotating-an-embedded-matplotlib-plot>
* <http://matplotlib.org/examples/user_interfaces/embedding_in_qt5.html>
* <https://www.mail-archive.com/matplotlib-users@lists.sourceforge.net/msg15322.html>

**Tests Run**

All tests that were run were only visual inspections of the output. There isn’t a way to lookup the expected answer or plot. I did try to visually compare a YouTube screenshot of an example shot, and match it up with the shot in the PGA Tour data, but that wasn’t always applicable or possible.

To test my results, I manually compared the sequential data for the X, Y, and Z coordinates with the visualization to see if the increase/decrease of data matched the visualization. For example, the Y coordinate represents the curvature from to right (or vice versa). I looked at the data of one shot at a time, and if the Y coordinate values were increasing, I checked to see if the visualization was plotting out to the right. Once the Y coordinate values were decreasing, I checked to see if the visualization was plotting back to the left. The same was applied to the X and Z coordinates, with X being the distance from the golfer, and Z being the height.

I then selected several shots with qualities seemed outrageous, or out of the norm. I looked at the data for the shot, and visualized the output in my head and then ran the program to see if it matched up. For example, Ollie Schneiderjans has a very low trajectory, so I picked out a couple of his shots were the Z coordinate did not increase much from the starting value. I then picked out another shot (this time from Smylie Kaufman), where the Z coordinate delta from start to apex was much larger. I expected a result where Ollie’s shots were barely above the bottom axis (X) compared to a high visualized path of Smylie’s shot. Once I ran the program, that is the output I got. I continued this exercise for roughly 20 more shots.

Additionally, I compared shots from different holes and tournaments, but found it to be quite inconsistent, due to the positioning of the TrackMan device and direction of the holes. A further improvement to the program would be to normalize the coordinates of every shot in the data so that they could be plotted together.

**Code Modifications**

From the sources above, I took the example solutions given, and then modified them to my own needs. There was no need for an improvement of the code used, but rather an implementation to my specific requirements. The code used from those sources was researched in a manner where it gave a generic implementation of a concept, rather than providing the program with the full, actual result. For example, I needed to find out how to add a matplotlib plot to a GUI, but once I found the code for the FIgureCanvas and Figure objects (which are built into the matplotlib.backends.backend\_qt5agg and matplotlib.figure modules), I modified the use of them to plot my own visualizations. I did not have to write the FIgureCanvas and Figure objects, as they are already included in matplotlib. I only have to research how to use these objects.

**Full Source Code**

**App.py**

import os  
import window\_main  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 os.chdir('C:\\Users\\ronri\\OneDrive\Mercyhurst\\DATA 520 - Intro to Programming\projects\\ideal-enigma')  
 app = window\_main.MainWindowLauncher()  
 app.\_\_init\_\_()

**window\_main.py**

# Load Libraries  
import os  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
# Load Resources  
import resource\_main  
  
# Load Dialogs  
import dialog\_about  
import dialog\_search  
  
  
class SplashWindow(QtWidgets.QMainWindow):  
 def \_\_init\_\_(self, parent=None):  
 super(SplashWindow, self).\_\_init\_\_(parent)  
  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(800, 600)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(0, 0, 800, 448))  
 self.label.setObjectName("label")  
 self.btnSelectFile = QtWidgets.QPushButton(self.centralwidget)  
 self.btnSelectFile.setGeometry(QtCore.QRect(30, 470, 201, 51))  
 self.btnSelectFile.setObjectName("btnSelectFile")  
 self.btnDemoFile = QtWidgets.QPushButton(self.centralwidget)  
 self.btnDemoFile.setGeometry(QtCore.QRect(30, 530, 201, 51))  
 self.btnDemoFile.setObjectName("btnDemoFile")  
 self.btnAbout = QtWidgets.QPushButton(self.centralwidget)  
 self.btnAbout.setGeometry(QtCore.QRect(570, 470, 201, 51))  
 self.btnAbout.setObjectName("btnAbout")  
 self.btnQuit = QtWidgets.QPushButton(self.centralwidget)  
 self.btnQuit.setGeometry(QtCore.QRect(570, 530, 201, 51))  
 self.btnQuit.setObjectName("btnQuit")  
 self.label\_2 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_2.setGeometry(QtCore.QRect(260, 470, 281, 101))  
 font = QtGui.QFont()  
 font.setPointSize(32)  
 font.setBold(True)  
 font.setWeight(75)  
 self.label\_2.setFont(font)  
 self.label\_2.setFrameShape(QtWidgets.QFrame.NoFrame)  
 self.label\_2.setAlignment(QtCore.Qt.AlignCenter)  
 self.label\_2.setObjectName("label\_2")  
 MainWindow.setCentralWidget(self.centralwidget)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 # Wire up Event Handlers  
 self.btnAbout.clicked.connect(self.show\_about\_dialog)  
 self.btnQuit.clicked.connect(self.close\_application)  
 self.btnDemoFile.clicked.connect(self.load\_demo\_file)  
 self.btnSelectFile.clicked.connect(self.select\_file)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "ProTracer"))  
 self.label.setText(\_translate("MainWindow", "<html><head/><body><p><img src=\":/logo/logo.jpg\"/></p></body></html>"))  
 self.btnSelectFile.setText(\_translate("MainWindow", "Select Data File"))  
 self.btnDemoFile.setText(\_translate("MainWindow", "Use Demo File"))  
 self.btnAbout.setText(\_translate("MainWindow", "About"))  
 self.btnQuit.setText(\_translate("MainWindow", "Quit"))  
 self.label\_2.setText(\_translate("MainWindow", "ProTracer"))  
  
 # Event Handlers  
 def show\_about\_dialog(self):  
 qtAboutDialog = QtWidgets.QDialog()  
 self.aboutDialog = dialog\_about.AboutDialog()  
 self.aboutDialog.setupUi(qtAboutDialog)  
 qtAboutDialog.exec()  
  
 def close\_application(self):  
 QtCore.QCoreApplication.quit()  
  
 def load\_demo\_file(self):  
 qtSearchDialog = QtWidgets.QDialog()  
 self.searchDialog = dialog\_search.SearchDialog()  
 self.searchDialog.setupUi(qtSearchDialog)  
 self.searchDialog.set\_filename(os.getcwd() + '\\data\\traj-detail-2017771908.TXT')  
 self.searchDialog.initialize()  
 qtSearchDialog.exec()  
  
 def select\_file(self):  
 options = QtWidgets.QFileDialog.Options()  
 # options |= QtWidgets.QFileDialog.DontUseNativeDialog  
 filename, \_ = QtWidgets.QFileDialog.getOpenFileName(  
 QtWidgets.QWidget(), "QFileDialog.getOpenFileName()", "",  
 "All Files (\*);;CSV Files (\*.csv);;Text Files (\*.txt)", options=options)  
  
 if filename:  
 qtSearchDialog = QtWidgets.QDialog()  
 self.searchDialog = dialog\_search.SearchDialog()  
 self.searchDialog.setupUi(qtSearchDialog)  
 self.searchDialog.set\_filename(filename)  
 self.searchDialog.initialize()  
 qtSearchDialog.exec()  
  
  
class MainWindowLauncher:  
 def \_\_init\_\_(self):  
 import sys  
  
 app = QtWidgets.QApplication(sys.argv)  
 app.setApplicationName('PyProTracer')  
  
 MainWindow = QtWidgets.QMainWindow()  
 main = SplashWindow()  
 main.setupUi(MainWindow)  
 MainWindow.show()  
  
 sys.exit(app.exec\_())

**dialog\_about.py**

from PyQt5 import QtCore, QtGui, QtWidgets  
  
import resource\_about  
  
class AboutDialog(object):  
 def setupUi(self, dlgAbout):  
 self.dialog = dlgAbout  
 dlgAbout.setObjectName("dlgAbout")  
 dlgAbout.resize(512, 273)  
 dlgAbout.setSizeGripEnabled(False)  
 dlgAbout.setModal(True)  
 self.label = QtWidgets.QLabel(dlgAbout)  
 self.label.setGeometry(QtCore.QRect(240, 10, 181, 51))  
 font = QtGui.QFont()  
 font.setPointSize(24)  
 self.label.setFont(font)  
 self.label.setObjectName("label")  
 self.label\_2 = QtWidgets.QLabel(dlgAbout)  
 self.label\_2.setGeometry(QtCore.QRect(240, 70, 111, 16))  
 self.label\_2.setObjectName("label\_2")  
 self.label\_3 = QtWidgets.QLabel(dlgAbout)  
 self.label\_3.setGeometry(QtCore.QRect(240, 100, 241, 31))  
 self.label\_3.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)  
 self.label\_3.setWordWrap(True)  
 self.label\_3.setObjectName("label\_3")  
 self.label\_4 = QtWidgets.QLabel(dlgAbout)  
 self.label\_4.setGeometry(QtCore.QRect(240, 140, 221, 21))  
 self.label\_4.setObjectName("label\_4")  
 self.btnCloseAbout = QtWidgets.QPushButton(dlgAbout)  
 self.btnCloseAbout.setGeometry(QtCore.QRect(300, 190, 111, 31))  
 self.btnCloseAbout.setObjectName("btnCloseAbout")  
 self.label\_5 = QtWidgets.QLabel(dlgAbout)  
 self.label\_5.setGeometry(QtCore.QRect(20, 20, 187, 225))  
 self.label\_5.setObjectName("label\_5")  
  
 self.retranslateUi(dlgAbout)  
 QtCore.QMetaObject.connectSlotsByName(dlgAbout)  
  
 # Wire up event handlers  
 self.btnCloseAbout.clicked.connect(self.close\_about\_dialog)  
  
 def retranslateUi(self, dlgAbout):  
 \_translate = QtCore.QCoreApplication.translate  
 dlgAbout.setWindowTitle(\_translate("dlgAbout", "About ProTracer"))  
 self.label.setText(\_translate("dlgAbout", "ProTracer"))  
 self.label\_2.setText(\_translate("dlgAbout", "Version: 1.0.0.0"))  
 self.label\_3.setText(\_translate("dlgAbout", "ProTracer is designed to visualize PGA Tour Trajectory data, mimicing television broadcasts."))  
 self.label\_4.setText(\_translate("dlgAbout", "Developed by Ron Richardson"))  
 self.btnCloseAbout.setText(\_translate("dlgAbout", "Close"))  
 self.label\_5.setText(\_translate("dlgAbout", "<html><head/><body><p><img src=\":/logo/logo-about.png\"/></p></body></html>"))  
  
  
 # Event Handlers  
 def close\_about\_dialog(self):  
 self.dialog.done(0)

**dialog\_search.py**

from PyQt5 import QtCore, QtGui, QtWidgets  
  
import os  
import pandas as pd  
import pt\_data  
import pt\_plot  
  
import widget\_mpl  
  
  
class SearchDialog(object):  
 def \_\_init\_\_(self):  
 self.filename = ''  
 self.data = pd.DataFrame()  
 self.pt = pt\_plot.ProTracerPlot()  
  
 def setupUi(self, dlgSearch):  
 self.dialog = dlgSearch  
  
 dlgSearch.setObjectName("dlgSearch")  
 dlgSearch.resize(1200, 800)  
 dlgSearch.setSizeGripEnabled(False)  
 # dlgSearch.setModal(True)  
  
 self.groupBox = QtWidgets.QGroupBox(dlgSearch)  
 self.groupBox.setGeometry(QtCore.QRect(10, 10, 1171, 351))  
 self.groupBox.setObjectName("groupBox")  
 self.btnAddShots = QtWidgets.QPushButton(self.groupBox)  
 self.btnAddShots.setGeometry(QtCore.QRect(10, 310, 141, 31))  
 self.btnAddShots.setObjectName("btnAddShots")  
 self.tblShotList = QtWidgets.QTreeWidget(self.groupBox)  
 self.tblShotList.setGeometry(QtCore.QRect(10, 20, 1151, 281))  
 self.tblShotList.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.tblShotList.setProperty("showDropIndicator", False)  
 self.tblShotList.setAlternatingRowColors(True)  
 self.tblShotList.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)  
 self.tblShotList.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)  
 self.tblShotList.setColumnCount(7)  
 self.tblShotList.setObjectName("tblShotList")  
 self.groupBox\_3 = QtWidgets.QGroupBox(dlgSearch)  
 self.groupBox\_3.setGeometry(QtCore.QRect(10, 370, 1171, 351))  
 self.groupBox\_3.setObjectName("groupBox\_3")  
 self.btnRemoveShots = QtWidgets.QPushButton(self.groupBox\_3)  
 self.btnRemoveShots.setGeometry(QtCore.QRect(10, 310, 141, 31))  
 self.btnRemoveShots.setObjectName("btnRemoveShots")  
 self.tblSelectedShots = QtWidgets.QTreeWidget(self.groupBox\_3)  
 self.tblSelectedShots.setGeometry(QtCore.QRect(10, 20, 1151, 281))  
 self.tblSelectedShots.setEditTriggers(QtWidgets.QAbstractItemView.NoEditTriggers)  
 self.tblSelectedShots.setProperty("showDropIndicator", False)  
 self.tblSelectedShots.setAlternatingRowColors(True)  
 self.tblSelectedShots.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)  
 self.tblSelectedShots.setSelectionBehavior(QtWidgets.QAbstractItemView.SelectRows)  
 self.tblSelectedShots.setColumnCount(7)  
 self.tblSelectedShots.setObjectName("tblSelectedShots")  
 self.btn2D = QtWidgets.QPushButton(dlgSearch)  
 self.btn2D.setGeometry(QtCore.QRect(410, 740, 181, 41))  
 self.btn2D.setObjectName("btn2D")  
 self.btn3D = QtWidgets.QPushButton(dlgSearch)  
 self.btn3D.setGeometry(QtCore.QRect(620, 740, 181, 41))  
 self.btn3D.setObjectName("btn3D")  
  
 self.tblShotList.header().setSectionResizeMode(QtWidgets.QHeaderView.ResizeToContents)  
 self.tblSelectedShots.header().setSectionResizeMode(QtWidgets.QHeaderView.ResizeToContents)  
  
 self.retranslateUi(dlgSearch)  
 QtCore.QMetaObject.connectSlotsByName(dlgSearch)  
  
 # Wire up event handlers  
 self.btnAddShots.clicked.connect(self.on\_add\_shots)  
 self.btnRemoveShots.clicked.connect(self.on\_remove\_shots)  
 self.btn2D.clicked.connect(self.on\_plot\_2d)  
 self.btn3D.clicked.connect(self.on\_plot\_3d)  
  
 def retranslateUi(self, dlgSearch):  
 \_translate = QtCore.QCoreApplication.translate  
 dlgSearch.setWindowTitle(\_translate("dlgSearch", "ProTracer Shot Search"))  
 self.groupBox.setTitle(\_translate("dlgSearch", "Shot List"))  
 self.btnAddShots.setText(\_translate("dlgSearch", "Add Selected Shots"))  
 self.tblShotList.setSortingEnabled(True)  
 self.tblShotList.headerItem().setText(0, \_translate("dlgSearch", "Player"))  
 self.tblShotList.headerItem().setText(1, \_translate("dlgSearch", "Tournament"))  
 self.tblShotList.headerItem().setText(2, \_translate("dlgSearch", "Round"))  
 self.tblShotList.headerItem().setText(3, \_translate("dlgSearch", "Hole"))  
 self.tblShotList.headerItem().setText(4, \_translate("dlgSearch", "Distance (yds)"))  
 self.tblShotList.headerItem().setText(5, \_translate("dlgSearch", "Apex (ft)"))  
 self.tblShotList.headerItem().setText(6, \_translate("dlgSearch", "Club Speed (mph)"))  
 self.tblShotList.headerItem().setText(7, \_translate("dlgSearch", "Ball Speed (mph)"))  
 self.tblSelectedShots.headerItem().setText(0, \_translate("dlgSearch", "Player"))  
 self.tblSelectedShots.headerItem().setText(1, \_translate("dlgSearch", "Tournament"))  
 self.tblSelectedShots.headerItem().setText(2, \_translate("dlgSearch", "Round"))  
 self.tblSelectedShots.headerItem().setText(3, \_translate("dlgSearch", "Hole"))  
 self.tblSelectedShots.headerItem().setText(4, \_translate("dlgSearch", "Distance (yds)"))  
 self.tblSelectedShots.headerItem().setText(5, \_translate("dlgSearch", "Apex (ft)"))  
 self.tblSelectedShots.headerItem().setText(6, \_translate("dlgSearch", "Club Speed (mph)"))  
 self.tblSelectedShots.headerItem().setText(7, \_translate("dlgSearch", "Ball Speed (mph)"))  
 self.groupBox\_3.setTitle(\_translate("dlgSearch", "Selected Shots"))  
 self.btnRemoveShots.setText(\_translate("dlgSearch", "Remove Selected Shots"))  
 self.tblSelectedShots.setSortingEnabled(True)  
 self.btn2D.setText(\_translate("dlgSearch", "ProTracer 2D"))  
 self.btn3D.setText(\_translate("dlgSearch", "ProTracer 3D"))  
  
 def set\_filename(self, filename):  
 self.filename = filename  
  
 def initialize(self):  
 if len(self.filename) > 0:  
 if os.path.isfile(self.filename):  
 self.data = pt\_data.load\_file(self.filename)  
 self.populate\_shot\_list()  
 else:  
 QtWidgets.QMessageBox.information(  
 QtWidgets.QWidget(), 'Error Loading Data File',  
 'The data file ({0}) could not be found. Please choose another file'.format(self.filename))  
 self.dialog.done(0)  
 else:  
 QtWidgets.QMessageBox.information(  
 QtWidgets.QWidget(), 'Error Loading Data File',  
 'No file was specified. Please select a file and try again.')  
 self.dialog.done(0)  
  
 def populate\_shot\_list(self):  
 shots = pt\_data.get\_all\_shots(self.data)  
 tree\_items = []  
 for i in range(len(shots)):  
 shot = shots.iloc[i]  
 item = [  
 str(shot["Player Last First"]),  
 str(shot["Tournament Name"]),  
 str(shot["Round"]),  
 str(shot["Hole Number"]),  
 str(round((shot["Total Distance"] / 12) / 3, 1)),  
 str(round(shot["Apex Height"], 1)),  
 str(round(shot["Club Head Speed"], 1)),  
 str(round(shot["Ball Speed"], 1)),  
 str(shot["Index"])  
 ]  
 tree\_items.append(QtWidgets.QTreeWidgetItem(item))  
 self.tblShotList.addTopLevelItems(tree\_items)  
  
 def on\_add\_shots(self):  
 items = self.tblShotList.selectedItems()  
  
 for item in items:  
 self.tblSelectedShots.addTopLevelItem(item.clone())  
 self.tblShotList.takeTopLevelItem(self.tblShotList.indexOfTopLevelItem(item))  
  
 self.tblShotList.clearSelection()  
  
 def on\_remove\_shots(self):  
 items = self.tblSelectedShots.selectedItems()  
  
 for item in items:  
 self.tblShotList.addTopLevelItem(item.clone())  
 self.tblSelectedShots.takeTopLevelItem(self.tblSelectedShots.indexOfTopLevelItem(item))  
  
 def get\_chosen\_shots(self):  
 shots = []  
 for i in range(self.tblSelectedShots.topLevelItemCount()):  
 shot = {  
 'Player Last First': self.tblSelectedShots.topLevelItem(i).data(0, 0),  
 'Tournament Name': self.tblSelectedShots.topLevelItem(i).data(1, 0),  
 'Round': self.tblSelectedShots.topLevelItem(i).data(2, 0),  
 'Hole Number': self.tblSelectedShots.topLevelItem(i).data(3, 0)  
 }  
 shots.append(shot)  
 return shots  
  
 def add\_shots\_to\_plot(self):  
 shots = []  
 for shot in self.get\_chosen\_shots():  
 shot\_data = pt\_data.get\_shot(self.data, shot["Player Last First"], shot["Tournament Name"],  
 shot["Round"], shot["Hole Number"])  
  
 summary = pt\_data.get\_shot\_summary(shot\_data)  
 shots.append((shot\_data, summary))  
  
 return shots  
  
 def on\_plot\_2d(self):  
 # self.move\_search\_dialog()  
 self.dialog.setGeometry(0, 100, self.dialog.width(), self.dialog.height())  
 self.ptdialog = widget\_mpl.ProTracerDialog(True)  
 self.ptdialog.set\_plot\_data(self.add\_shots\_to\_plot())  
 self.ptdialog.on\_draw\_2d()  
 self.ptdialog.exec()  
  
 def on\_plot\_3d(self):  
 self.dialog.setGeometry(0, 100, self.dialog.width(), self.dialog.height())  
 self.ptdialog = widget\_mpl.ProTracerDialog(False)  
 self.ptdialog.set\_plot\_data(self.add\_shots\_to\_plot())  
 self.ptdialog.on\_draw\_3d()  
 self.ptdialog.exec()

**widget\_mpl.py**

# http://scipy-cookbook.readthedocs.io/items/Matplotlib\_Qt\_with\_IPython\_and\_Designer.html  
# https://github.com/eliben/code-for-blog/blob/master/2009/qt\_mpl\_bars.py  
# https://pythonspot.com/en/pyqt5-matplotlib/  
# https://stackoverflow.com/questions/3972158/how-to-plot-on-my-gui  
  
# https://stackoverflow.com/questions/29357442/example-of-embedding-matplotlib-in-pyqt5  
# 3d toolbar options not supported  
  
from PyQt5 import QtCore, QtGui, QtWidgets  
import matplotlib.pyplot as plt  
import mpl\_toolkits.mplot3d.axes3d as p3  
import matplotlib.animation as animation  
import numpy as np  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
from matplotlib.backends.backend\_qt5agg import NavigationToolbar2QT as NavigationToolbar  
from matplotlib.figure import Figure  
  
import matplotlib  
matplotlib.use('QT5Agg')  
  
  
class PlotCanvas(FigureCanvas):  
 def \_\_init\_\_(self, parent=None, width=8, height=6, dpi=100):  
 fig = Figure(figsize=(width, height), dpi=dpi)  
 self.axes = fig.add\_subplot(111)  
  
 FigureCanvas.\_\_init\_\_(self, fig)  
 self.setParent(parent)  
  
 FigureCanvas.setSizePolicy(self,  
 QtWidgets.QSizePolicy.Expanding,  
 QtWidgets.QSizePolicy.Expanding)  
 FigureCanvas.updateGeometry(self)  
  
  
class PlotCanvas3D(FigureCanvas):  
 def \_\_init\_\_(self, parent=None, width=6, height=8, dpi=100):  
 fig = Figure(figsize=(width, height), dpi=dpi)  
 self.axes = fig.add\_subplot(111, projection='3d')  
  
 FigureCanvas.\_\_init\_\_(self, fig)  
 self.setParent(parent)  
  
 FigureCanvas.setSizePolicy(self,  
 QtWidgets.QSizePolicy.Expanding,  
 QtWidgets.QSizePolicy.Expanding)  
 FigureCanvas.updateGeometry(self)  
  
  
class ProTracerDialog(QtWidgets.QDialog):  
 def \_\_init\_\_(self, is2d=True, parent=None):  
 QtWidgets.QDialog.\_\_init\_\_(self, parent)  
 self.setWindowTitle("ProTracer")  
 # self.resize(800, 600)  
  
 if is2d:  
 self.resize(1500, 500)  
 self.canvas = PlotCanvas(self, width=15, height=5)  
 else:  
 self.resize(500, 1000)  
 self.canvas = PlotCanvas(self, width=5, height=10)  
  
 screen = QtWidgets.QDesktopWidget().availableGeometry()  
 self.setGeometry(screen.width() - self.width(), 75, self.width(), self.height())  
  
 self.padding = 25.0  
 self.linewidth = 8  
 self.interval = 50  
 self.aspect\_size = 3  
 self.aspect\_ratio = 0.0  
  
 self.data = []  
 self.labels = []  
 self.extrapolated = []  
 self.lines = []  
  
 self.xmax = -1  
 self.ymax = -1  
 self.zmax = -1  
  
 def set\_plot\_data(self, shots):  
 self.data = []  
 self.shots = shots  
  
 def on\_draw\_2d(self):  
 # set up toolbar for 2D only  
 self.canvas.toolbar = NavigationToolbar(self.canvas, self.canvas)  
 self.canvas.toolbar.update()  
  
 for i in range(len(self.shots)):  
 self.add\_plot\_data(self.shots[i][0], self.shots[i][1], True)  
  
 self.plot\_2d()  
  
 def on\_draw\_3d(self):  
 for i in range(len(self.shots)):  
 self.add\_plot\_data(self.shots[i][0], self.shots[i][1], False)  
  
 self.plot\_3d()  
  
 def adjust\_coordinates(self, L, is2d=True):  
 # assumes first value is lowest for plot  
 # trim list when x coord hits 0 after 1st?  
 output = []  
 if is2d:  
 minval = L[0]  
 else:  
 minval = min(L)  
  
 for item in L:  
 output.append(item - minval)  
  
 return output  
  
 def add\_plot\_data(self, shot\_data, shot\_summary, is2d=True, include\_extrapolated=False):  
 if include\_extrapolated:  
 x = shot\_data["Trajectory X Coordinate"].tolist()  
 y = shot\_data["Trajectory Y Coordinate"].tolist()  
 z = shot\_data["Trajectory Z Coordinate"].tolist()  
 else:  
 x = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory X Coordinate"].tolist()  
  
 y = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory Y Coordinate"].tolist()  
  
 z = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory Z Coordinate"].tolist()  
  
 # Unnecessary at this point  
 extrapolated = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Extrapolated"].tolist()  
  
 x = self.adjust\_coordinates(x, is2d)  
 y = self.adjust\_coordinates(y, is2d)  
 z = self.adjust\_coordinates(z, is2d)  
  
 self.xmax = max(self.xmax, max(x))  
 self.ymax = max(self.ymax, max(y))  
 self.zmax = max(self.zmax, max(z))  
  
 self.data.append(np.array((x, y, z)))  
 self.labels.append(shot\_summary)  
 self.extrapolated.append(extrapolated)  
  
 def init\_2d(self):  
 self.canvas.ax.clear()  
 self.canvas.ax.set\_xlim(0, self.xmax + self.padding)  
 self.canvas.ax.set\_ylim(0, self.zmax + self.padding)  
  
 for i in range(len(self.data)):  
 # self.lines[i].set\_data([], [])  
 x = self.data[i][0, 0:1]  
 y = self.data[i][2, 0:1]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, linewidth=self.linewidth, label=label)  
  
 self.canvas.draw()  
  
 def update\_2d\_lines(self, num, datas, lines):  
 self.canvas.ax.clear()  
 for i in range(len(self.data)):  
 x = self.data[i][0, :num]  
 y = self.data[i][2, :num]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, linewidth=self.linewidth, label=label)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()  
  
 def plot\_2d(self):  
 self.xmax += self.padding  
 self.ymax += self.padding  
 self.zmax += self.padding  
 self.aspect\_ratio = self.xmax // float(self.zmax)  
  
 self.canvas.figure = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 self.canvas.ax = plt.axes(xlim=(0, self.xmax), ylim=(0, self.zmax))  
 self.canvas.ax.get\_xaxis().set\_visible(False)  
 self.canvas.ax.get\_yaxis().set\_visible(False)  
  
 longest = -1  
 for i in range(len(self.data)):  
 longest = max(longest, self.data[i].shape[1])  
  
 ani = animation.FuncAnimation(self.canvas.figure, self.update\_2d\_lines, longest, fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False, init\_func=self.init\_2d)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()  
  
 def init\_3d(self):  
 self.canvas.ax.clear()  
 self.canvas.ax.mouse\_init()  
 self.canvas.ax.set\_xlim3d([0, self.xmax + self.padding])  
 self.canvas.ax.set\_ylim3d([0, self.ymax + self.padding])  
 self.canvas.ax.set\_zlim3d([0, self.zmax + self.padding])  
  
 for i in range(len(self.data)):  
 x = self.data[i][0, 0:1]  
 y = self.data[i][1, 0:1]  
 z = self.data[i][2, 0:1]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, z, linewidth=self.linewidth, label=label)  
  
 self.canvas.draw()  
  
 def update\_3d\_lines(self, num, datas, lines):  
 self.canvas.ax.clear()  
 self.canvas.ax.mouse\_init()  
 for i in range(len(self.data)):  
 x = self.data[i][0, :num]  
 y = self.data[i][1, :num]  
 z = self.data[i][2, :num]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, z, linewidth=self.linewidth, label=label)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
  
 self.canvas.ax.set\_xlim3d([0, self.xmax + self.padding])  
 self.canvas.ax.set\_ylim3d([0, self.ymax + self.padding])  
 self.canvas.ax.set\_zlim3d([0, self.zmax + self.padding])  
  
 self.canvas.ax.\_axis3don = False  
 self.canvas.ax.set\_axis\_off()  
  
 self.canvas.draw()  
  
 def plot\_3d(self):  
 self.xmax += self.padding  
 self.ymax += self.padding  
 self.zmax += self.padding  
 self.aspect\_ratio = self.xmax // float(self.zmax)  
  
 self.canvas.figure = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 self.canvas.ax = p3.Axes3D(self.canvas.figure)  
 self.canvas.ax.view\_init(elev=0, azim=45)  
 self.canvas.ax.\_axis3don = False  
 self.canvas.ax.set\_axis\_off()  
  
 longest = -1  
 for i in range(len(self.data)):  
 longest = max(longest, self.data[i].shape[1])  
  
 ani = animation.FuncAnimation(self.canvas.figure, self.update\_3d\_lines, longest, fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()

**pt\_data.py**

import pandas as pd  
import numpy as np  
  
  
def load\_file(filename, sep=';'):  
 *""" (string) -> DataFrame  
  
 Returns a pandas DataFrame with all the contents of the file specified.  
 """* data = pd.read\_csv(filename, sep=sep)  
  
 # Create helper columns  
 data["Player Full Name"] = data["Player First Name"] + " " + data["Player Last Name"]  
 data["Player Last First"] = data["Player Last Name"] + ", " + data["Player First Name"]  
 data["Index"] = range(data.shape[0])  
  
 return data  
  
  
def get\_all\_golfers(data, last\_name\_first=True):  
 *""" (DataFrame, bool) -> list  
  
 Returns a list of all unique golfers in the data.  
 """* if last\_name\_first:  
 return sorted(data["Player Last First"].unique().tolist())  
 else:  
 return sorted(data['Player Full Name'].unique().tolist())  
  
  
def get\_golfers\_by\_tournament(data, tournament, last\_name\_first=True):  
 *""" (DataFrame, string) -> list  
  
 Returns a list of unique golfers for a certain tournament.  
 """* if last\_name\_first:  
 return data.loc[(data["Tournament Name"] == tournament)]["Player Last First"].unique().tolist()  
 else:  
 return data.loc[(data["Tournament Name"] == tournament)]["Player Full Name"].unique().tolist()  
  
  
def get\_all\_tournaments(data):  
 *""" (DataFrame) -> list  
  
 Returns a list of all unique tournaments in the data.  
 """* return data['Tournament Name'].unique().tolist()  
  
  
def get\_tournaments\_by\_golfer(data, golfer, last\_name\_first=True):  
 *""" (DataFrame, string, bool) -> list  
  
 Returns a list of all the unique tournaments for a certain golfer.  
 """* if last\_name\_first:  
 return data.loc[(data["Player Last First"] == golfer)]["Tournament Name"].unique().tolist()  
 else:  
 return data.loc[(data["Player Full Name"] == golfer)]["Tournament Name"].unique().tolist()  
  
  
def get\_shots\_by\_golfer\_tournament(data, golfer, tournament, last\_name\_first=True):  
 *""" (DataFrame, string, string, bool) -> DataFrame  
   
 Returns a DataFrame of all shots by a golfer for a tournament.  
 """* # *TODO: put into helper function* if last\_name\_first:  
 col = "Player Last First"  
 else:  
 col = "Player Full Name"  
  
 return data.loc[  
 (data[col] == golfer) &  
 (data["Tournament Name"] == tournament) &  
 (data['Trajectory Sequence'] == 1)  
 ]  
  
  
def get\_all\_shots(data):  
 *""" (DataFrame) -> DataFrame  
  
 Returns a DataFrame of all unique shots in the data file.  
 """* return data.loc[  
 (data["Trajectory Sequence"] == 1)  
 ].sort\_values(by=['Player Last Name', 'Player First Name', 'Tournament Name', 'Round', 'Hole Number'])  
  
  
def get\_shot(data, golfer, tournament, round, hole, last\_name\_first=True):  
 *""" (DataFrame, string, string, int, int, bool) -> DataFrame  
  
 Returns the full trajectory data for a shot.  
 """* if last\_name\_first:  
 col = "Player Last First"  
 else:  
 col = "Player Full Name"  
  
 return data.loc[  
 (data[col] == golfer) &  
 (data["Tournament Name"] == tournament) &  
 (data["Round"] == int(round)) &  
 (data["Hole Number"] == int(hole))  
 ].sort\_values(by=['Trajectory Sequence'])  
  
  
def get\_shot\_summary(shot\_data):  
 *""" (DataFrame) -> dictionary  
  
 Returns a dictionary of summary data.  
 """* summary = {}  
 summary["Player First Name"] = shot\_data.iloc[0]["Player First Name"]  
 summary["Player Last Name"] = shot\_data.iloc[0]["Player Last Name"]  
 summary["Player Full Name"] = shot\_data.iloc[0]["Player Full Name"]  
 summary["Tournament Name"] = shot\_data.iloc[0]["Tournament Name"]  
 summary["Round"] = shot\_data.iloc[0]["Round"]  
 summary["Hole Number"] = shot\_data.iloc[0]["Hole Number"]  
 summary["Club Head Speed"] = shot\_data.iloc[0]["Club Head Speed"]  
 summary["Ball Speed"] = shot\_data.iloc[0]["Ball Speed"]  
 summary["Smash Factor"] = shot\_data.iloc[0]["Smash Factor"]  
 summary["Vertical Launch Angle"] = shot\_data.iloc[0]["Vertical Launch Angle"]  
 summary["Apex Height"] = shot\_data.iloc[0]["Apex Height"]  
 summary["Actual Flight Time"] = shot\_data.iloc[0]["Actual Flight Time"]  
 summary["Actual Range"] = shot\_data.iloc[0]["Actual Range"]  
 summary["Actual Height"] = shot\_data.iloc[0]["Actual Height"]  
 summary["Distance of Impact"] = shot\_data.iloc[0]["Distance of Impact"]  
 summary["Club"] = shot\_data.iloc[0]["Club"]  
 summary["Total Distance"] = shot\_data.iloc[0]["Total Distance"]  
 summary["Ending Location Description"] = shot\_data.iloc[0]["Ending Location Description"]  
 summary["Weather"] = shot\_data.iloc[0]["Weather"]  
  
 return summary

**pt\_plot.py**

import numpy as np  
import matplotlib.pyplot as plt  
import mpl\_toolkits.mplot3d.axes3d as p3  
import matplotlib.animation as animation  
  
  
class ProTracerPlot:  
 def \_\_init\_\_(self, add\_extrapolation=False, is\_2d=True, canvas=None):  
 self.padding = 25.0  
 self.linewidth = 8  
 self.interval = 50  
 self.aspect\_size = 3  
 self.aspect\_ratio = 0.0  
  
 self.data = []  
 self.labels = []  
 self.extrapolated = []  
 self.lines = []  
  
 self.xmax = -1  
 self.ymax = -1  
 self.zmax = -1  
  
 self.extrapolation = add\_extrapolation  
 self.is\_2d = is\_2d  
  
 self.canvas = canvas  
  
 def adjust\_coordinates(self, L):  
 # assumes first value is lowest for plot  
 # trim list when x coord hits 0 after 1st?  
 output = []  
 if self.is\_2d:  
 minval = L[0]  
 else:  
 minval = min(L)  
  
 for item in L:  
 output.append(item - minval)  
  
 return output  
  
 def add\_plot\_data(self, shot\_data, shot\_summary):  
 if self.extrapolation:  
 x = shot\_data["Trajectory X Coordinate"].tolist()  
 y = shot\_data["Trajectory Y Coordinate"].tolist()  
 z = shot\_data["Trajectory Z Coordinate"].tolist()  
 else:  
 x = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory X Coordinate"].tolist()  
  
 y = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory Y Coordinate"].tolist()  
  
 z = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Trajectory Z Coordinate"].tolist()  
  
 # Unnecessary at this point  
 extrapolated = shot\_data.loc[  
 (shot\_data['Extrapolated'] == 'N')  
 ]["Extrapolated"].tolist()  
  
 x = self.adjust\_coordinates(x)  
 y = self.adjust\_coordinates(y)  
 z = self.adjust\_coordinates(z)  
  
 self.xmax = max(self.xmax, max(x))  
 self.ymax = max(self.ymax, max(y))  
 self.zmax = max(self.zmax, max(z))  
  
 self.data.append(np.array((x, y, z)))  
 self.labels.append(shot\_summary)  
 self.extrapolated.append(extrapolated)  
  
 def init\_2d(self):  
 if self.canvas is None:  
 for i in range(len(self.data)):  
 self.lines[i].set\_data([], [])  
 return self.lines  
 else:  
 self.canvas.ax.clear()  
 self.canvas.ax.set\_xlim(0, self.xmax + self.padding)  
 self.canvas.ax.set\_ylim(0, self.zmax + self.padding)  
  
 for i in range(len(self.data)):  
 x = self.data[i][0, 0:1]  
 y = self.data[i][2, 0:1]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, linewidth=self.linewidth, label=label)  
  
 self.canvas.draw()  
  
 def update\_2d\_lines(self, num, datas, lines):  
 if self.canvas is None:  
 for i in range(len(datas)):  
 self.lines[i].set\_data(datas[i][0, :num], datas[i][2, :num])  
 return self.lines  
 else:  
 self.canvas.ax.clear()  
 for i in range(len(self.data)):  
 x = self.data[i][0, :num]  
 y = self.data[i][2, :num]  
  
 print(x)  
 print(y)  
 print('\n\n')  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, linewidth=self.linewidth, label=label)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()  
  
 def plot\_2d(self, title="ProTracer 2D"):  
 self.is\_2d = True  
 self.xmax += self.padding  
 self.ymax += self.padding  
 self.zmax += self.padding  
 self.aspect\_ratio = self.xmax // float(self.zmax)  
  
 print(type(self.canvas))  
 if self.canvas is None:  
 print('Doing normal plot')  
 fig = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 ax = plt.axes(xlim=(0, self.xmax), ylim=(0, self.zmax))  
 ax.set\_title(title)  
  
 for i in range(len(self.data)):  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
 self.lines.append(ax.plot(self.data[i][0, 0:1], self.data[i][2, 0:1], linewidth=self.linewidth, label=label)[0])  
  
 ani = animation.FuncAnimation(fig, self.update\_2d\_lines, fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False, init\_func=self.init\_2d)  
  
 handles, labels = ax.get\_legend\_handles\_labels()  
 ax.legend(handles, labels)  
  
 mng = plt.get\_current\_fig\_manager()  
 mng.window.showMaximized()  
  
 plt.show()  
 else:  
 print('Canvas plot')  
 self.canvas.figure = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 self.canvas.ax = plt.axes(xlim=(0, self.xmax), ylim=(0, self.zmax))  
 self.canvas.ax.get\_xaxis().set\_visible(False)  
 self.canvas.ax.get\_yaxis().set\_visible(False)  
  
 longest = -1  
 for i in range(len(self.data)):  
 longest = max(longest, self.data[i].shape[1])  
  
 ani = animation.FuncAnimation(self.canvas.figure, self.update\_2d\_lines, longest,  
 fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False, init\_func=self.init\_2d)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()  
  
 def init\_3d(self):  
 if self.canvas is None:  
 for i in range(len(self.data)):  
 self.lines[i].set\_data([], [], [])  
 return self.lines  
 else:  
 self.canvas.ax.clear()  
 self.canvas.ax.set\_xlim3d([0, self.xmax + self.padding])  
 self.canvas.ax.set\_ylim3d([0, self.ymax + self.padding])  
 self.canvas.ax.set\_zlim3d([0, self.zmax + self.padding])  
  
 for i in range(len(self.data)):  
 x = self.data[i][0, 0:1]  
 y = self.data[i][1, 0:1]  
 z = self.data[i][2, 0:1]  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, z, linewidth=self.linewidth, label=label)  
  
 self.canvas.draw()  
  
 def update\_3d\_lines(self, num, datas, lines):  
 if self.canvas is None:  
 for i in range(len(datas)):  
 lines[i].set\_data(datas[i][0:2, :num])  
 lines[i].set\_3d\_properties(datas[i][2, :num])  
 return lines  
 else:  
 self.canvas.ax.clear()  
 self.canvas.ax.mouse\_init()  
 for i in range(len(self.data)):  
 x = self.data[i][0, :num]  
 y = self.data[i][1, :num]  
 z = self.data[i][2, :num]  
  
 print(x)  
 print(y)  
 print(z)  
 print('\n\n')  
  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.canvas.ax.plot(x, y, z, linewidth=self.linewidth, label=label)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
  
 self.canvas.ax.set\_xlim3d([0, self.xmax + self.padding])  
 self.canvas.ax.set\_ylim3d([0, self.ymax + self.padding])  
 self.canvas.ax.set\_zlim3d([0, self.zmax + self.padding])  
  
 self.canvas.ax.\_axis3don = False  
 self.canvas.ax.set\_axis\_off()  
  
 self.canvas.draw()  
  
 def plot\_3d(self, azim=45, axis\_on=False, title="ProTracer 3D"):  
 self.is\_2d = False  
 self.xmax += self.padding  
 self.ymax += self.padding  
 self.zmax += self.padding  
 self.aspect\_ratio = self.xmax // float(self.zmax)  
  
 if self.canvas is None:  
 fig = plt.figure()  
 ax = p3.Axes3D(fig)  
 ax.set\_title(title)  
 ax.view\_init(elev=0, azim=azim)  
 ax.\_axis3don = axis\_on  
  
 for i in range(len(self.data)):  
 label = '{0} - Round {1}'.format(  
 self.labels[i]["Player Last Name"],  
 self.labels[i]["Round"])  
  
 self.lines.append(  
 ax.plot(self.data[i][0, 0:1], self.data[i][1, 0:1], self.data[i][2, 0:1], linewidth=self.linewidth,  
 label=label)[0])  
  
 ax.set\_xlim3d([0, self.xmax + self.padding])  
 ax.set\_label('x')  
  
 ax.set\_ylim3d([0, self.ymax + self.padding])  
 ax.set\_label('y')  
  
 ax.set\_zlim3d([0, self.zmax + self.padding])  
 ax.set\_label('z')  
  
 ani = animation.FuncAnimation(fig, self.update\_3d\_lines, self.data[0].shape[1], fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False)  
  
 handles, labels = ax.get\_legend\_handles\_labels()  
 ax.legend(handles, labels)  
 # plt.axis('off')  
  
 mng = plt.get\_current\_fig\_manager()  
 mng.window.showMaximized()  
  
 plt.show()  
 else:  
 self.canvas.figure = plt.figure(figsize=(self.aspect\_ratio \* self.aspect\_size, self.aspect\_size))  
 self.canvas.ax = p3.Axes3D(self.canvas.figure)  
 self.canvas.ax.view\_init(elev=0, azim=45)  
 self.canvas.ax.\_axis3don = False  
 self.canvas.ax.set\_axis\_off()  
  
 longest = -1  
 for i in range(len(self.data)):  
 longest = max(longest, self.data[i].shape[1])  
  
 ani = animation.FuncAnimation(self.canvas.figure, self.update\_3d\_lines, longest,  
 fargs=(self.data, self.lines),  
 interval=self.interval, blit=False, repeat=False)  
  
 handles, labels = self.canvas.ax.get\_legend\_handles\_labels()  
 self.canvas.ax.legend(handles, labels)  
 self.canvas.draw()