(1) 程式介面說明：

將 exe 檔與藥讀取的 txt 檔放到同一個資料夾，執行 exe 檔。

然後在訓練過程，會印出每 epch 時的 MSE。

最後會印出車與軌道的圖，並且上面有各個數值。

(2) 程式碼說明

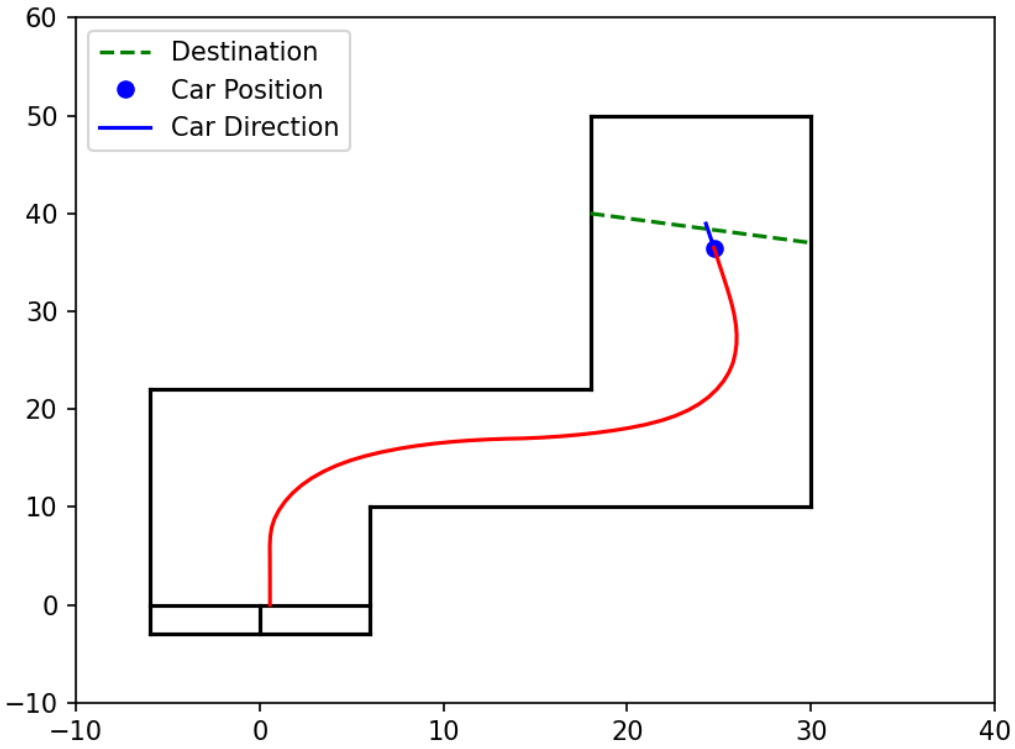
基本上就是建立一個 RBFN 網路。去預測 angle 然後在 playground 的 train function 然後呼叫 RBFN 網路訓練 txt 檔，將 model 用來預測每一個 action。

```
class RBFN:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01, n_iters=1000, sigma=1.0):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.sigma = sigma    # RBF kernel's width parameter

        # 初始化权重和偏置
        self.weights = np.random.randn(hidden_size, output_size)
        self.bias = np.zeros(output_size)

    def initialize_centers_with_kmeans(self, X):
        # 使用 K-Means 找到隐藏层中心
        kmeans = KMeans(n_clusters=self.hidden_size, random_state=0).fit(X)
        self.centers = kmeans.cluster_centers_

    def rbf_kernel(self, X, centers):
        # 高斯核函数计算输入点到各中心点的距离
        return np.exp(-cdist(X, centers, 'sqeuclidean') / (2 * self.sigma ** 2))

    def fit(self, X, y):
        # 使用 K-Means 初始化中心
        self.initialize_centers_with_kmeans(X)

        for epoch in range(self.n_iters):
            # 计算 RBF 层输出（高斯核值）
```

```
            hidden_output = self.rbf_kernel(X, self.centers)

            # 计算输出层预测值
            output = hidden_output.dot(self.weights) + self.bias

            # 计算误差
            error = output - y

            # 更新权重和偏置（简单的梯度下降）
            self.weights -= self.learning_rate * hidden_output.T.dot(error) / len(X)
            self.bias -= self.learning_rate * np.sum(error) / len(X)

            # 每 100 次迭代打印一次误差
            if epoch % 100 == 0:
                mse = np.mean(error ** 2)
                print(f"Epoch {epoch}: MSE = {mse:.4f}")

    def predict(self, X):
        # 计算 RBF 层输出（高斯核值）
        hidden_output = self.rbf_kernel(X, self.centers)

        # 计算输出层预测值
        return hidden_output.dot(self.weights) + self.bias

    def calculate_mse(self, X, y):
        # 计算均方误差
        predictions = self.predict(X)
        mse = np.mean((predictions - y) ** 2)
        return mse
```

(3) 實驗結果

```
left_dist: 12.717627875933296
front_dist: 9.516957344917664
right_dist: 7.991763947432066
Car X Position: 24.7248192026759
Car Y Position: 36.494002202907915
```



4. 分析：

在參數調整部分：

學習率：大約在 0.01 ~ 0.03 嘗試，太大的話，會梯度爆炸，得到 nan

隱藏層數目：大約設置 100，如果太大一樣會 train 不出來，太小容易 underfitting 。

訓練次數：一般來說訓練越多 MSE 會越低，但是有時也會發現 MSE 雖然很低，但是跑出來的結果很爛，應該是 overfitting 了。

另外由於類神經網路需要把資料標準化，可以用 Z-score 和歸一化，後發現 Z-score 效果稍好。