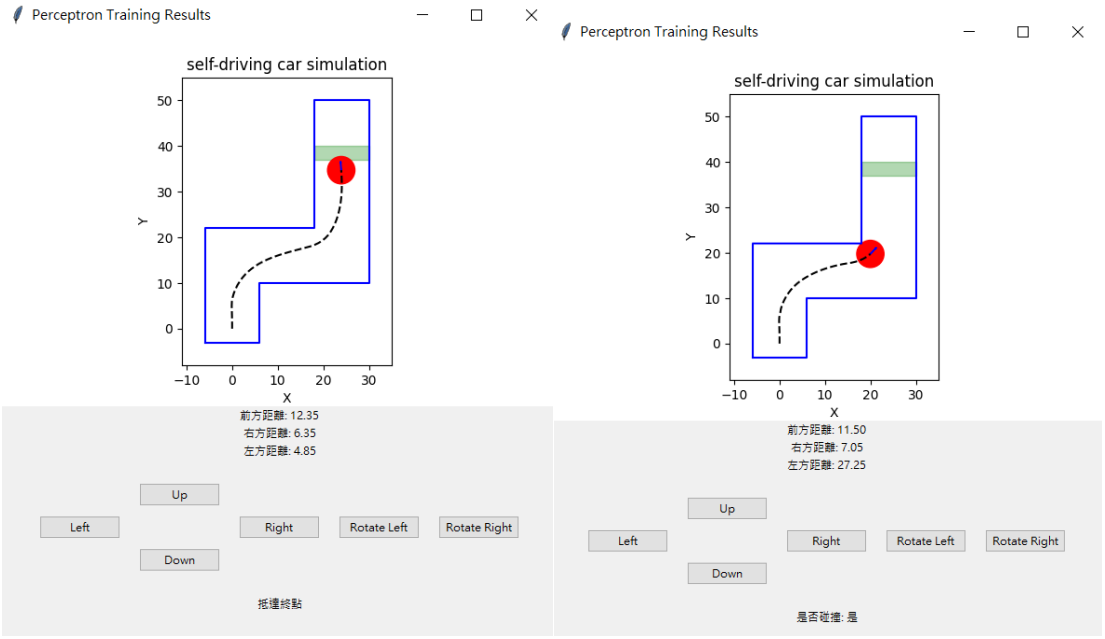
註:我符合加分資格，有自行編寫模擬程式，並未使用範例程式

(一) 介面介紹:

左為成功到終點，右為失敗撞牆

打開 exe 檔會先後台進行訓練約 5-10 秒，訓練完畢會直接打開 GUI 開始跑程式 0.5 秒更新一次畫面

因為模擬程式是自己寫的，故我加了方向鍵跟控制旋轉的(可以介入當前再播的動畫)



(二) 程式碼說明

Math_tool.py:

數學運算用 class，裡面全是 static function

MLPnetwork.py:

MLP 模型，初始化參數可以調整 4D/6D

裡面包含 load_data、train、和依據輸入產生下一個方向盤的 state

這邊我是先將輸入正規化後，把正規化後的輸出(-1 ~ 1)直接*40(-40 ~ 40)

Self_driving_car.py:

- `load_data(self)`: 讀取軌道座標和終點區域的資料。
- `update_state(self)`: 根據下圖產生下一個 state

$$x(t+1) = x(t) + \cos[\phi(t) + \theta(t)] + \sin[\theta(t)]\sin[\phi(t)] \qquad (10.18)$$

$$y(t+1) = y(t) + \sin[\phi(t) + \theta(t)] - \sin[\theta(t)]\cos[\phi(t)] \qquad (10.19)$$

$$\phi(t+1) = \phi(t) - \arcsin[\frac{2\sin[\theta(t)]}{b}] \qquad (10.20)$$

- `calculate_distances(self)`: 計算車體前方、左方、右方的距離。
- `calculate_distance_in_direction(self, angle)`: 根據給定的角度計算車體在該方向上的距離。用有點土法煉鋼的方法，所以距離超過 100 就會變 inf 且效能不大好以 0.05 為一基本單位來計算
- `check_collision(self)`: 檢查車體是否與任何牆壁發生碰撞。
- `reach_goal(self)`: 檢查車體是否抵達目標終點。但其實就是檢查車體與終點的 rectangle 發生碰撞與否
- `check_car_collision(self, target)`: 檢查車體是否與指定目標發生碰撞。檢查方法為判斷線段(拉兩個連續的頂點)與圓(車體)有沒有相交

Gui.py(4D/6D 沒差很多，但我拆兩個方便 debug):
- `__init__(self)`: 初始化 GUI 窗口，包括模擬軌跡、車輛、感測器等元素。

畫圖相關函式
- `draw_map(self)`: 繪製地圖，包含軌道和終點區域。
- `draw_car(self)`: 繪製車輛，包括車子外圓和朝向箭頭。
- `draw_fig(self)`: 繪製整個模擬圖形，包括地圖和車輛。
- `draw_track_trace(self)`: 繪製軌跡追蹤線，展示車輛運動軌跡。
- `fig_setting(self)`: 設定圖形顯示屬性，如坐標軸範圍、標題等。
- `create_simulation_figure(self)`: 創建模擬圖形的主窗口。

模擬相關函式
- `start_simulation(self)`: 啟動模擬運行的主循環。
- `run_simulation(self)`: 模擬主循環，更新車輛狀態並實時更新 GUI。

畫面更新相關函式
- `update_gui(self)`: 更新 GUI 元素，包括軌跡、車輛、距離感測器顯示。
- `update_distances_labels(self)`: 更新感測器距離的顯示文字。

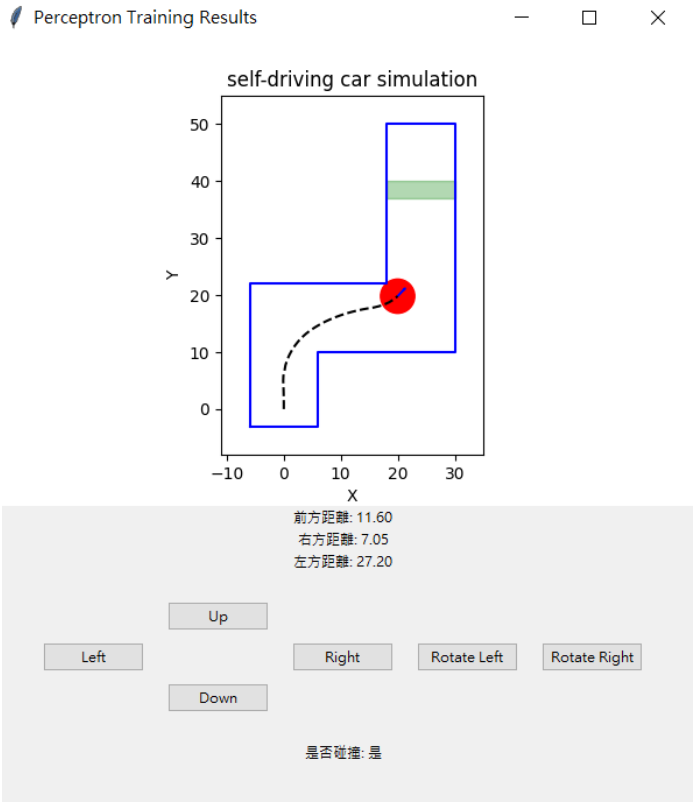
碰撞顯示相關函式
- `create_sensor(self)`: 創建感測器顯示區域，展示前、右、左方距離。
- `create_collision_label(self)`: 創建碰撞標籤，顯示車輛是否碰撞。
- `update_collision_label(self)`: 更新碰撞標籤的文字內容。
- `create_buttons(self)`: 創建控制按鈕，包括上、下、左、右移動和旋轉按鈕。

方向鍵相關函式
- `move_car(self, dx, dy)`: 移動車輛，根據給定的 dx 和 dy 值。

- `rotate_car(self, angle)`: 旋轉車輛，根據給定的角度。
- `move_up(self)`: 向上移動車輛的操作函數。
- `move_down(self)`: 向下移動車輛的操作函數。
- `move_left(self)`: 向左移動車輛的操作函數。
- `move_right(self)`: 向右移動車輛的操作函數。
- `rotate_left(self)`: 左轉車頭的操作函數。
- `rotate_right(self)`: 右轉車頭的操作函數。

(三) 實驗結果與分析

4D 跟 6D 有顯著的成功率差異，6D 成功率高很多，但兩者都不是百分百成功，且發現失敗的案例通常會在下圖的位置撞到



推測試資料樣本不足，軌道內的蠻多點都沒有 case 的，模型就只能沿著主要路徑走，但會因為訓練的部分隨機因素而撞牆

下圖為將 train6D 的每個點印在圖上的結果