Hw1: Path Planning

- A*

  在 A*的演算法實作當中，我使用了 python standard library 的 queue 當中的 PriorityQueue 作為儲存待搜尋節點的資料結構。其中，key 為 node 的(x, y)坐標，value 是該 node 的 F 值，也就是 path cost(g) + heuristic estimate(h)的值。

  演算法架構上，最外層的 loop 是判斷 PriorityQueue 中是否還有 node 沒有探索，如果還有，則每次取出一個 F 值最小的 node 進行探索。在探索前會先檢查該 node 是否距離 goal 足夠的近(dist<=inter)，如果是則令該 node 為 goal_node 便可以結束 loop，若否則會將其加入 visited list，確保不會重復探索，並且開始八個方向的探索。

  在探索過程中，每一個探索到的 node 都需檢查是否超出邊界、是否會發生碰撞、以及是否造訪過，如果皆否，則可以進一步判斷該 node 是否已經在 PriorityQueue 當中等待被取出進行探索，如果已經在 queue 內，則需要檢查此時的路徑是否可以讓該 node 的 F 值更小，代表這條路徑到 Goal 的 cost 更小，如果是則更新 queue 內的 F 值，並重新指派 parent，如果否則不將這個 node 重複加入 queue 當中，避免後續重複探索。

  至此，便完成了整個 A*演算法的實作。

- RRT*

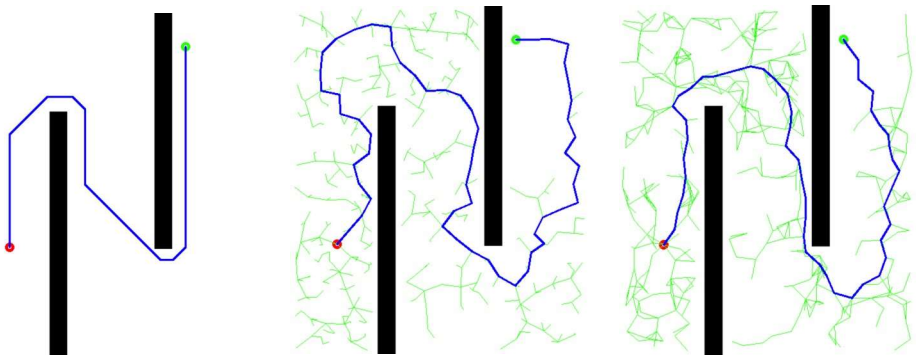
  在 RRT*演算法當中，因為大部分是基於 RRT 進行改進，所以我們僅需新增 Re-Parent 和 Re-Wire 這兩個 function 即可。

  在 Re-Parent 當中，我們首先需要設定一個範圍，僅在這個範圍做 Re-Parent，範圍設定為半徑 inter 並以該 node 為圓心的圓形。隨後，遍歷這個範圍內的其他所有 node，找到一個可以從其他 node 到該 node 的最短的路徑，並將該 node 的 parent 重新設定成這個最短路徑的 node。

  而在 Re-Wire 當中，與 Re-Parent 的做法很相似，同樣在該範圍內檢查該 node 到範圍內其他所有點是否可以取得更短的路徑，如果有更短的路徑存在，則讓被更新的 node 的 parent 設定為該 node。

在每次 RRT 演算法取得新的 node 時都去執行以上兩步，可以讓整體的路徑更加平滑一些，並且可能可以找到更短的路徑。

- Result



A*　　RRT　　RRT*

從結果圖可以看出，A*的路徑最短，而 RRT*對比起 RRT 來說，路徑更加平滑了一些，並且在分支的探索上也可以看得出 Re-Parent 和 Re-Wire 的存在。