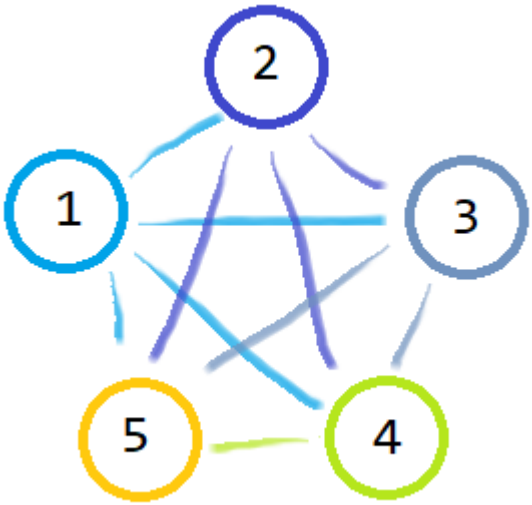
許多遊戲都會需要碰撞檢測來判斷兩物體的碰撞，但這些演算法通常是較為昂貴的操作，如果無法有效率的選擇檢測目標，很可能會大幅降低執行的速度。

在之前的”多邊形碰撞檢測”文中，也有提到當檢測物體越來越多時，基本逐一檢測的效率會越來越差，複雜度約為 O((n-1)^2)，就算排除重複檢測過的物體，也要 O(S(n-1))次(S 為等差級數總和)，不管怎樣，逐一檢測的方法一定會走訪所有物件。

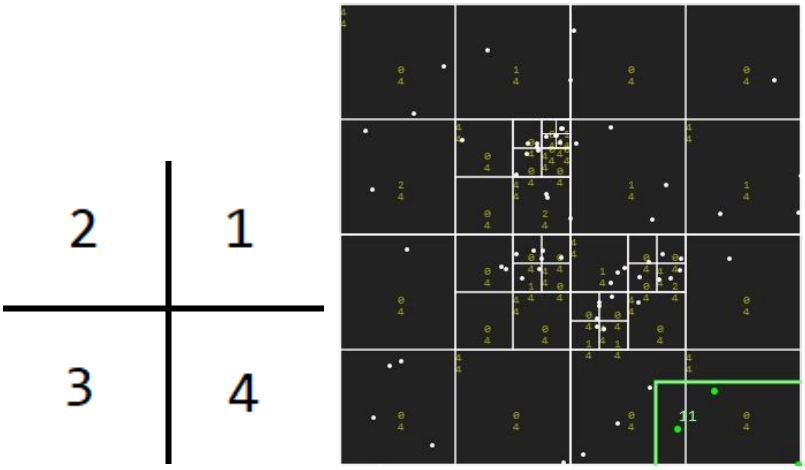


從上圖可以得知直接全部檢測的話是 4 x 5 = 20，如果扣掉重複，至少是 4 + 3 + 2 + 1 = 10，，但真正的問題是，**就算 1 和 3 距離這麼遠，也會照樣檢查它們**，那有沒有方法能解決這種狀況呢?

這就是文章的主要內容，QuadTree。

什麼是 QuadTree?

四叉樹是一種劃分 2D 區域的樹狀資料結構，類似一般的二元樹，但子節點是 4 個，而不是 2 個。區域的劃分架構類似這樣 ：

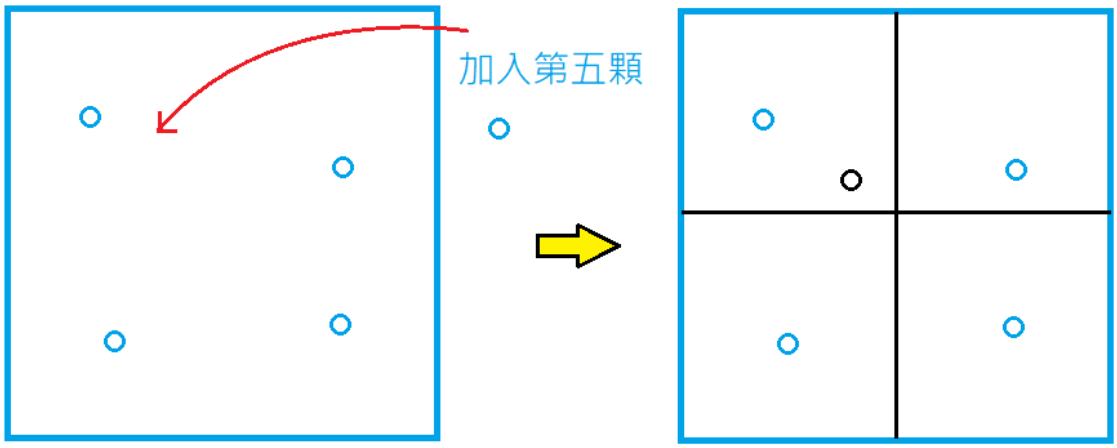


並限制每個區域能容納的上限，當超過後就將該區域再往下分割 4 塊，這樣就能夠將每個物體進行區域分類，這樣在檢察的時候就可以鎖定部分區域的物體，從而增加效率。

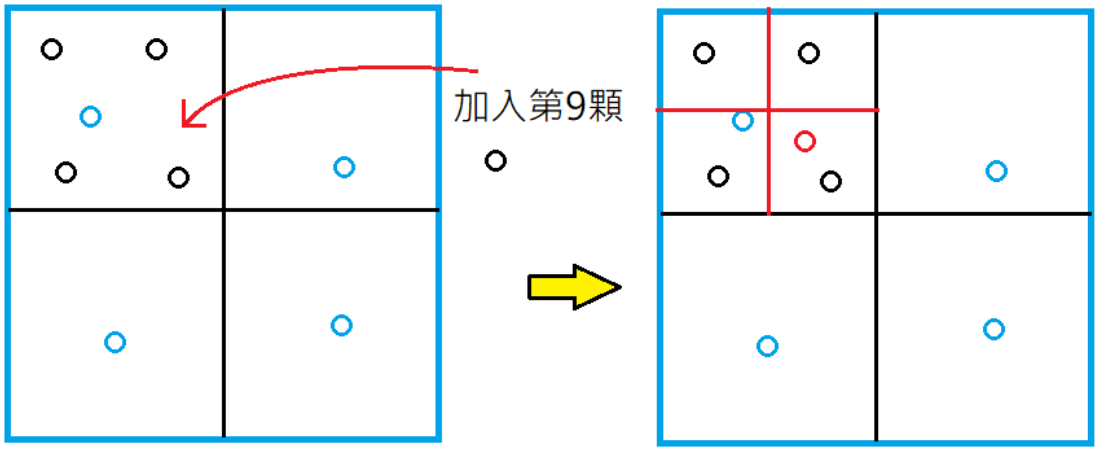
插入流程:

假設我先設定每個區域只能容納 4 個物體，只要超過該容量，就要分割該區域。

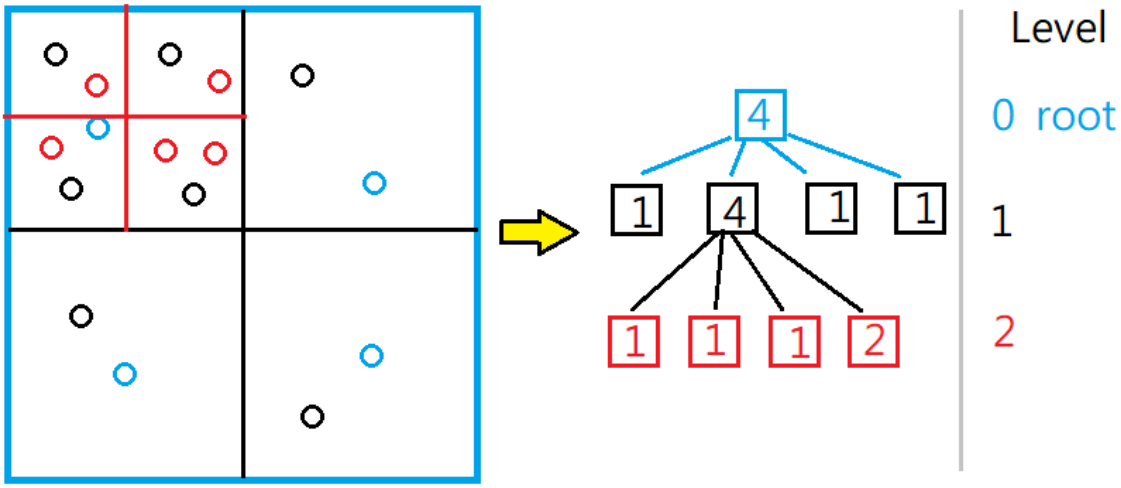
左圖方形區域已經有 4 個物體，想再加入第 5 個時，就必須分割成 4 個子區域，再將第 5 顆分類到最近的左上角區域中，如右圖:



以此類推，當要放入第 9 顆物體時，發現黑色區域也滿了，所以就再往下分割紅色區域，並放入離該物體最近的右下角區域中，如下圖:



最後，就可以得出這樣的樹狀結構，這就是為什麼會叫做 4 叉樹的原因，如下圖:

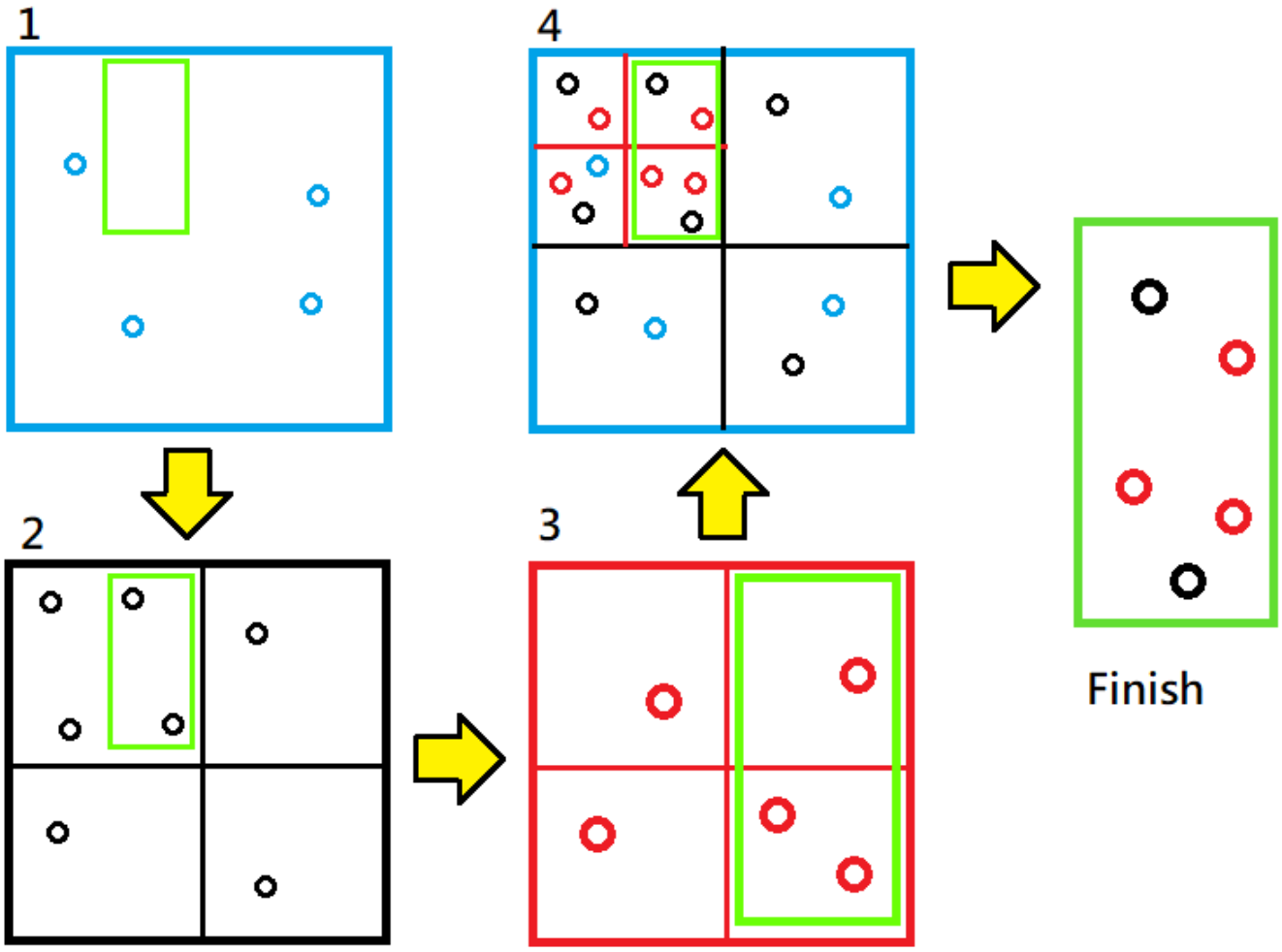


搜尋流程:

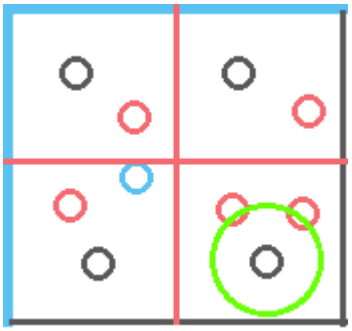
給定一搜尋範圍，並逐一排除不可能的區域，最後取得搜尋範圍內的物體。

以下圖為例，假設我要檢測綠色框框內的物體是否發生碰撞，步驟如下：

1. 與藍框區域作碰撞檢測，發現有所交集，檢測藍色區域的物體是否包含在綠框中，發現並沒有。
2. 與藍框的黑色子區域作檢測，發現只與左上的區域有所交集，所以排除另外三個區域，並發現有 2 個物體包含在綠框內。
3. 再往下檢察左上黑框的紅色子區域，並排除左上、左下，發現有 3 個物體在綠框內，紅框沒有子領域，走訪結束。
4. 最後回傳搜尋到的 5 個物體。



如果是碰撞檢測的應用的話，就是把綠框換成該物體周圍可能發生碰撞的範圍，就在套入搜尋，就可以更加簡化碰撞檢測的流程，如下圖:



假設要檢查有機會與黑色物體碰撞的物體，透過上述的篩選流程，最後只要針對與綠框相交的 2 個紅色物體，來執行碰撞檢測，以此達成碰撞效率優化。

這差不多就是四叉樹的原理。