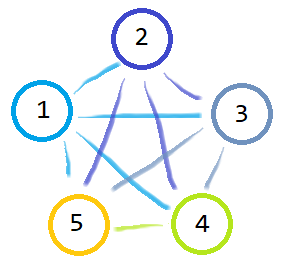
許多遊戲都會需要碰撞檢測來判斷兩物體的碰撞，但這些演算法通常是較為昂貴的操作，如果無法有效率的選擇檢測目標，很可能會大幅降低執行的速度。

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^2)，就算排除重複檢測過的物體，只是是逐一檢測的方法就一定會走訪所有物件。



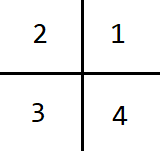
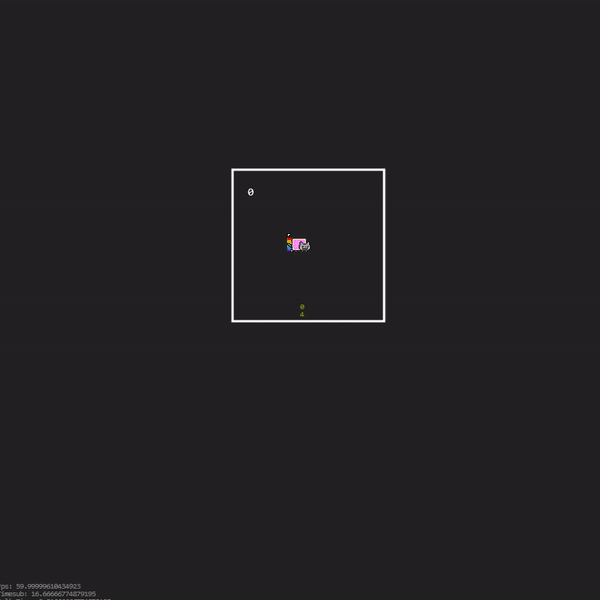
從上圖可以得知直接全部檢測的話是4 x 5 = 20，如果扣掉重複，至少是4 + 3 + 2 + 1 = 10，，但真正的問題是，**就算1和3距離這麼遠，也會照樣檢查它們**，那有沒有方法能解決這種狀況呢?

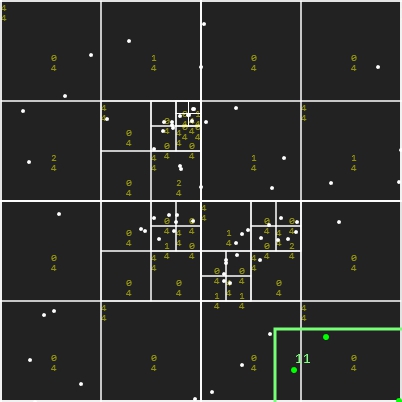
這就是文章的主要內容，QuadTree。

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 :

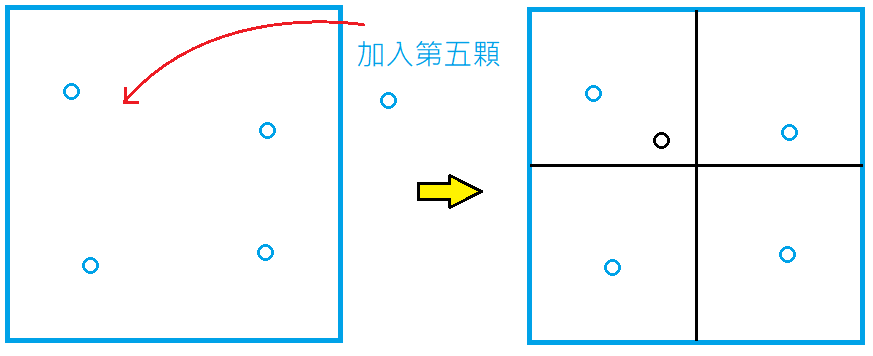


並限制每個區域能容納的上限，當超過後就將該區域再往下分割4塊，這樣就能夠將每個物體進行區域分類，這樣在檢察的時候就可以鎖定部分區域的物體，從而增加效率。

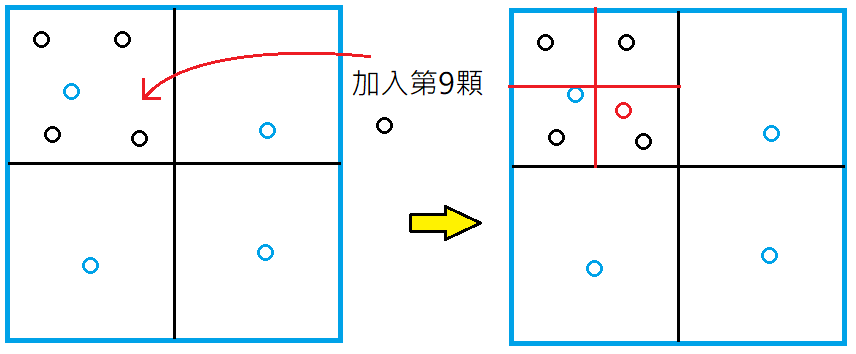
插入流程:

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。

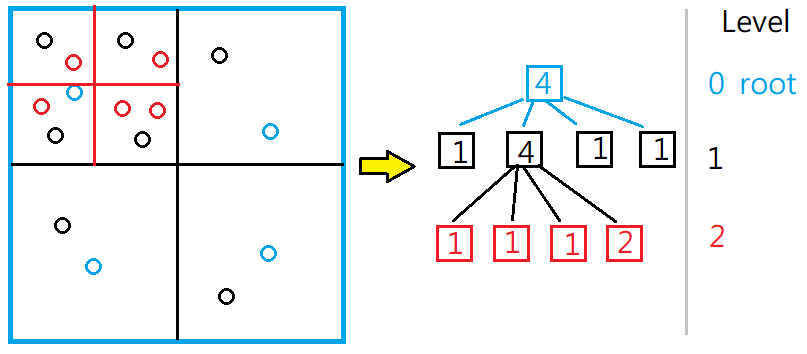
左圖方形區域已經有4個物體，想再加入第5個時，就必須分割成4個子區域，再將第5顆分類到最近的左上角區域中，如右圖:



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:

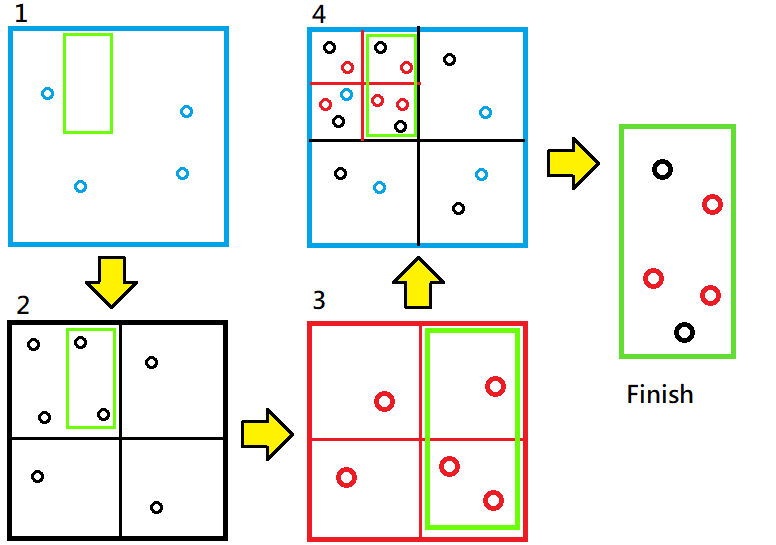


搜尋流程:

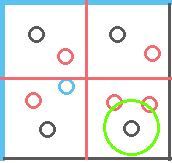
接續前面插入的狀態，並逐一排除不可能的區域，最後取得搜尋範圍內的物體。

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，再來套入搜尋，就能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。

這就是四叉樹的原理。

如何實作QuadTree:

整理大致的插入順序:

1. 如果該點不屬於A區域，結束
2. 如果該點屬於A區域，且A區域容量還夠的時候，直接將該點加入A區域
3. 如果A區域容量不夠且沒有分割時，將A區域劃分4個子區域，並將該點加入離它最近的子區域

首先，建立Point與Rect物件

………CODE……………..